Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros Industriales

Dynamic Response Optimization of Vehicles through Efficient Multibody Formulations and Automatic Differentiation Techniques

Doctoral Thesis

ALFONSO CALLEJO GOENA
Industrial Engineer

Madrid, 2013

Departamento de Ingeniería Mecánica y Fabricación
Escuela Técnica Superior de Ingenieros Industriales

Dynamic Response Optimization of Vehicles through Efficient Multibody Formulations and Automatic Differentiation Techniques

Doctoral Thesis

ALFONSO CALLEJO GOENA
Industrial Engineer

Advisor
Javier García de Jalón de la Fuente
Dr. Ingeniero Industrial
Catedrático de Matemática Aplicada

Madrid, 2013

Acknowledgements

Out of the many people that have helped me out along the way, I would like to express my special gratitude to my supervisor, Javier García de Jalón, who has taught me, among many other things, the art of debugging; to Francisco Aparicio, who welcomed me very warmly at the research institute from the first day; and to my family and friends, who have always supported me.

I would also like to thank Santiago Tapia, Pablo Luque, Andrés Hidalgo, Wilmar Cabrera, Michele Macchi, Daniel Arribas, Ángel Martín, Francisco Badea, Eduardo Elipe, María Dolores Gutiérrez, Luis Ormazábal, Yongjun Pan, Kshitij Kulshreshtha, Krishna Narayanan, John McPhee and Tom Uchida for their feedback and collaboration, as well as the whole crew from the Instituto Universitario de Investigación del Automóvil (INSIA).

Finally, I acknowledge the funding from the Education Department of the Government of Navarra, without which this Thesis would not exist, and the help from the Spanish Ministry of Science and Innovation via research projects.

To you all, I offer my sincere gratitude. I am more now than before I met you.

Agradecimientos

De las numerosas personas que me han ayudado en este largo camino, me gustaría dar gracias especialmente a mi director de tesis, Javier García de Jalón, que me ha enseñado, entre otras muchas cosas, el arte de la depuración; a Francisco Aparicio, que me acogió cálidamente en el centro de investigación desde el primer día; y a mi familia y amigos, que siempre han estado ahí apoyándome.

También quiero dar gracias a Santiago Tapia, Pablo Luque, Andrés Hidalgo, Wilmar Cabrera, Michele Macchi, Daniel Arribas, Ángel Martín, Francisco Badea, Eduardo Elipe, María Dolores Gutiérrez, Luis Ormazábal, Yongjun Pan, Kshitij Kulshreshtha, Krishna Narayanan, John McPhee y Tom Uchida por su cooperación y respaldo, así como a todo el personal del Instituto Universitario de Investigación del Automóvil (INSIA).

Por último, agradezco también la financiación del Gobierno de Navarra, sin la cual esta Tesis no sería realidad, así como las ayudas del Ministerio de Ciencia e Innovación a través de proyectos de investigación.

A todos, gracias de verdad. Soy más desde que os conozco.

Abstract

Each day, the design and development of vehicle suspension systems relies more on computer-aided design and computer-aided engineering tools, which allow anticipating the problems and solving them ahead of time. Dynamic behavior and characteristics are thus simulated accurately and inexpensively with moderate computational times and resources. There is, however, an iterative component in the process, which involves the manual definition of designs in a trial-and-error manner. This Thesis takes a step towards the development of an efficient simulation framework capable of simulating, analyzing and evaluating vehicle suspension designs, and automatically improving them by varying the design parameters towards the optimal solution.

The multibody systems approach is hereby used to model a three-dimensional 18-degrees-of-freedom coach in a comprehensive yet efficient way. The suspension geometry and characteristics resemble the ones from the real vehicle, as do the rest of vehicle parameters. In order to simulate vehicle dynamics, an efficient, state-of-the-art multibody formulation based on Maggi's equations is employed, and a three-dimensional graphics viewer is developed. As a result, vehicle maneuvers can be simulated faster than real-time.

Once the dynamics are ready, a sensitivity analysis is crucial for a robust optimization. To that end, a mathematical technique is introduced, which allows differentiating the dynamic variables within the multibody formulation in a general, algorithmic, accurate to machine precision, and reasonably efficient way: automatic differentiation. This method propagates the derivatives with respect to the design parameters throughout the computer code, with little user interaction. In contrast with other attempts in the literature, mostly not general-purpose, a benchmarking of libraries is carried out, a hybrid direct-automatic differentiation approach for the computation of sensitivities is developed, and several real-life examples are analyzed.

Finally, a design optimization process of the aforementioned vehicle is carried out. Four different types of dynamic response optimization are presented: parameter identification, handling optimization, ride comfort optimization and multi-objective optimization; all of which are applied to the design of the coach example. Together with analytical and visual proof of the results, efficiency considerations are made. In summary, the dynamic behavior of vehicles is improved by using the multibody systems approach, along with advanced differentiation and optimization techniques, enabling an automatic, accurate and efficient tuning of design parameters.

Resumen

El diseño y desarrollo de sistemas de suspensión para vehículos se basa cada día más en el diseño por ordenador y en herramientas de análisis por ordenador, las cuales permiten anticipar problemas y resolverlos por adelantado. El comportamiento y las características dinámicas se calculan con precisión, bajo coste, y recursos y tiempos de cálculo reducidos. Sin embargo, existe una componente iterativa en el proceso, que requiere la definición manual de diseños a través de técnicas "prueba y error". Esta Tesis da un paso hacia el desarrollo de un entorno de simulación eficiente capaz de simular, analizar y evaluar diseños de suspensiones vehiculares, y de mejorarlos hacia la solución óptima mediante la modificación de los parámetros de diseño.

La modelización mediante sistemas multicuerpo se utiliza aquí para desarrollar un modelo de autocar con 18 grados de libertad, de manera detallada y eficiente. La geometría y demás características de la suspensión se ajustan a las del vehículo real, así como los demás parámetros del modelo. Para simular la dinámica vehicular, se utiliza una formulación multicuerpo moderna y eficiente basada en las ecuaciones de Maggi, a la que se ha incorporado un visor 3D. Así, se consigue simular maniobras vehiculares en tiempos inferiores al tiempo real.

Una vez que la dinámica está disponible, los análisis de sensibilidad son cruciales para una optimización robusta y eficiente. Para ello, se presenta una técnica matemática que permite derivar las variables dinámicas dentro de la formulación, de forma algorítmica, general, con la precisión de la máquina, y razonablemente eficiente: la diferenciación automática. Este método propaga las derivadas con respecto a las variables de diseño a través del código informático y con poca intervención del usuario. En contraste con otros enfoques en la bibliografía, generalmente particulares y limitados, se realiza una comparación de librerías, se desarrolla una formulación híbrida directa-automática para el cálculo de sensibilidades, y se presentan varios ejemplos reales.

Finalmente, se lleva a cabo la optimización de la respuesta dinámica del vehículo citado. Se analizan cuatro tipos distintos de optimización: identificación de parámetros, optimización de la maniobrabilidad, optimización del confort y optimización multi-objetivo, todos ellos aplicados al diseño del autocar. Además de resultados analíticos y gráficos, se incluyen algunas consideraciones acerca de la eficiencia. En resumen, se mejora el comportamiento dinámico de vehículos por medio de modelos multicuerpo y de técnicas de diferenciación automática y optimización avanzadas, posibilitando un ajuste automático, preciso y eficiente de los parámetros de diseño.

To my parents and siblings.

For a mechanic, you seem to do
an excessive amount of thinking...

C3PO

Contents

Chapter 1

Introduction

The objective of improving the dynamic response of vehicles can obviously be achieved in different ways. For instance, active suspension systems can be implemented in the real vehicle in order to control pitch and roll during severe maneuvers. Another option is to tune suspension parameters once the vehicle is manufactured, which is expensive, time-consuming and does not allow for important changes. Nowadays, design engineering is imperative in vehicle production, so it is applied more and more each day. The shortening of development and manufacturing processes along with the fast renovation of vehicle models has led to an increase in demand of reliable tools for the design of new models. Thus, the ideal approach would be to adjust suspension parameters to as large as possible in the design stage by running and optimizing virtual vehicles. This is the problem faced by vehicle dynamic response optimization.

Over the last few decades, vehicle safety has boosted the development of virtual testing tools, allowing thousands of tests to be run before the vehicle is manufactured. These tools have an important role in the calculation of the structure and in the suspension design. The former is crucial for crashworthiness and fatigue life, while the latter is crucial for dynamic response and comfort. However, virtual testing tools are tricky and have their drawbacks. Computer models need to be validated in order to be physically meaningful, and the validation process often requires on-track tests and model tuning, which can all be very time consuming. Moreover, the great variety of modeling approaches, objectives and implementations makes it difficult to generalize modeling results.

The main objective of this Thesis is to improve the dynamic performance of vehicles in the design stage. To that end, vehicles are modeled using the multibody systems approach in a way that vehicle dynamics can be simulated efficiently and accurately. Then, state-of-the-art optimization techniques are applied to suspension parameters in order to improve the dynamic behavior. In contrast with other fields of computational mechanics where optimization techniques are very widespread, the dynamic response optimization of multibody

systems shows lights and shadows in the literature, which reveals the complexity of the numerical problem at hand. Some insight will be provided here, in an attempt to enable an all-inclusive general-purpose evaluation and improvement of the vehicle dynamic response without any further simplifications or assumptions, besides multibody systems theory.

Thus, several fields of mechanics, computer science and mathematics meet in the problem under study, among others: vehicle dynamics, multibody dynamics, automatic differentiation, sensitivity analysis, mathematical optimization and computer programming. It is therefore crucial to tackle each aspect with an appropriate angle and depth. In this section, the fundamental preliminaries of vehicle dynamics, multibody dynamics and optimization are summarized.

1.1 Vehicle dynamics

Road vehicles are studied in this Thesis from a global dynamics point of view. Mechanics of materials are thus considered less important for the global dynamic response than the inertia properties of the parts, the interaction between them and the forces generated by the contact with the road.

When real-life systems are optimized, the optimization is only as good as the model, because no matter how good an optimization method is, a rough model is going to lead to unrealistic results. Among the different qualities of a vehicle model, the dynamic behavior of a vehicle is influenced, above all, by the following aspects, of which some can be modified by the designer:

– Definition of front and rear axle geometry
– Variation of the geometry with suspension motion
– Stiffness and damping of the suspension system
– Tire contact forces

When using multibody dynamics to simulate vehicles, the axle geometry is the actual one, thus having the accuracy of the measurement system. The spring, damper and tire models are obviously very important, determining the fidelity of the model. The main advantage of multibody systems with respect to other approximated methods is that the variation of the geometry with the suspension movement is inherent to the suspension topology, thus no special curves or tables are required, as it is exact.

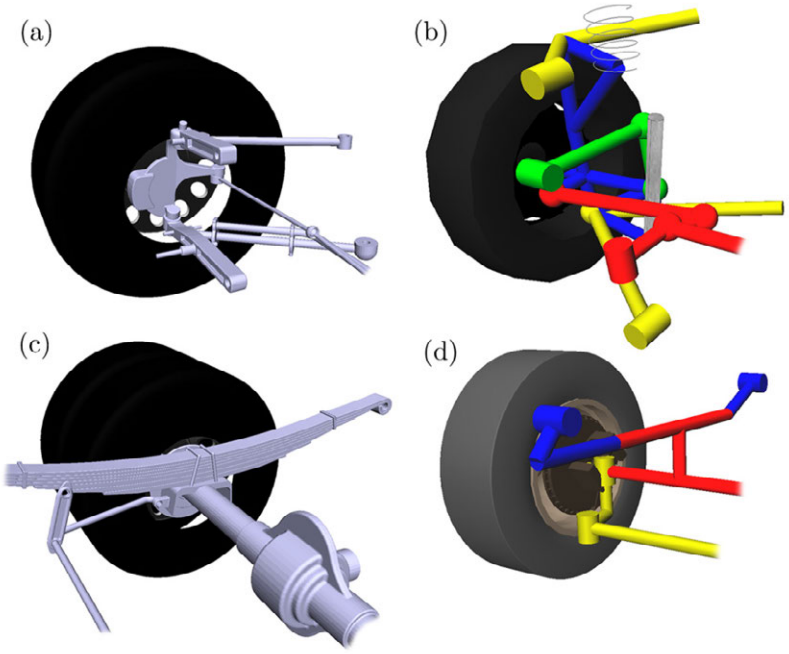


Figure 1.1: Independent ((a) and (b)) and dependent ((c) and (d)) suspension systems.

There are two basic types of suspension systems: *dependent* and *independent*. Dependent (or solid axle) ones are the simplest way of attaching a pair of wheels to the vehicle, by mounting them on a solid axle. The axle must be attached to the body in a way that the relative vertical displacement and rotation around the longitudinal axis are allowed. The movement of a wheel is thus dependent on the movement of the other wheel. The second type of suspension systems are independent suspensions. They allow each wheel to move up and down independently of the opposite wheel. Among independent suspensions, double wishbone, McPherson strut and multilink are the most common types. See Figure 1.1 for four examples of suspension systems.

The position and orientation of the bodywork (or chassis) is crucial in the analysis of the dynamic behavior. Here, the global (inertial) reference frame, XYZ, is placed on the ground under the front axle. Initially, the global X-direction is longitudinal with respect to the vehicle, the Y-direction is lateral, and the Z-direction is vertical (see Figure 1.2). The rotations are measured around a non-inertial reference frame, xyz, located on the COG of the vehicle, whose axes are initially parallel to the global ones. These rotations will be referred to as *roll* φ (x-axis), *pitch* θ (y-axis) and *yaw* ψ (z-axis) throughout this Thesis.

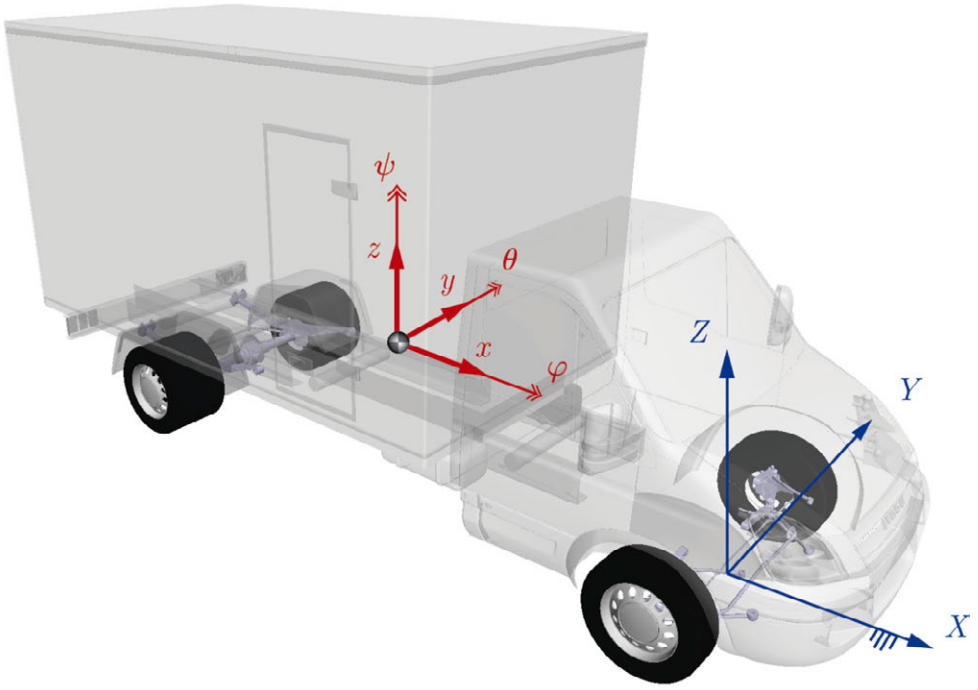


Figure 1.2: Position and orientation of the vehicle chassis.

Wheels need to be accurately modeled as well if the multibody model is used in real-life vehicle dynamics applications. The position and orientation of the wheels is usually determined by the geometry, position and orientation of the uprights. The *toe* angle is the angle between the wheel and a longitudinal plane measured from a plan view of the vehicle. It is positive when the wheels tend towards the front and negative in the opposite case. From a static configuration to a motion one, drive wheels tend to increase their toe angle, whereas trailer wheels tend to decrease it. See Figure 1.3 for wheel angle depictions.

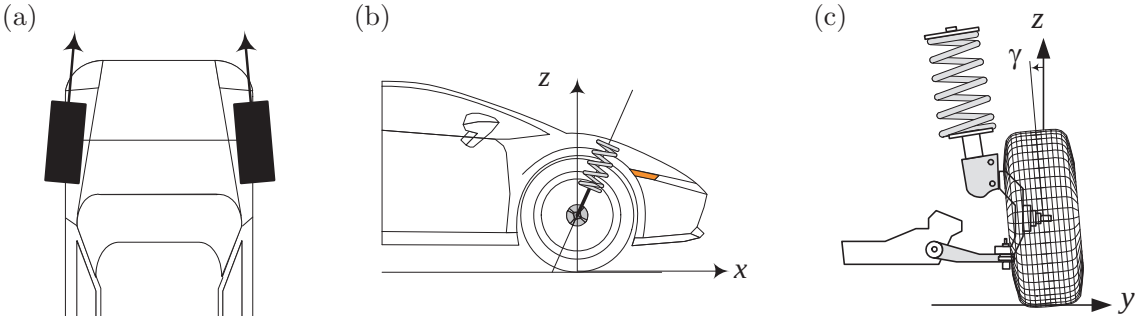


Figure 1.3: Wheel angles: (a) toe, (b) caster and (c) camber.
With kind permission of Springer Science+Business Media (Jazar, 2008).

The *caster* angle is the angle between the kingpin (or pivot) axis and the vertical axis, measured from the side view. If the intersection between the pivot axis and the road is ahead of the contact patch, the caster angle is positive, and vice versa. An important effect of the caster angle is the self-centering behavior of

the steering system, making it easier to negotiate turns and travel in a straight line. The caster angle increases when braking and decreases with traction.

Finally, the *camber* angle is measured between a vertical plane and the plane containing the wheel, from a front view. The camber angle affects the shape of the contact patch between the tire and the ground. It is positive when the upper part of the wheel is farther from the chassis than the lower one, and vice versa. The camber angle can reduce the effect of bumps on the steering. Positive angles tend to increase axle side slip, and are affected by the vehicle roll angle.

1.2 Dynamic simulation of multibody systems

The multibody dynamics approach is used here for the accurate simulation of vehicle dynamics. In the literature, this approach has proven to be an efficient yet accurate way of simulating vehicle dynamics while capturing the most important kinematic and dynamic characteristics.

Multibody systems are mechanical systems made up of rigid and/or flexible bodies interconnected by rigid and/or flexible joints, subject to external forces, and which generally undergo large displacements. Kinematic joints are defined as rigid connections between bodies, which enable certain relative degrees of freedom (DOFs) and constrain others. Examples of multibody systems are robots, artificial satellites, vehicles, machines, and even granular materials. The field of multibody system dynamics belongs to the areas of computer-aided design (CAD), computer-aided engineering (CAE) and computational mechanics.

Note that the analysis of multibody systems is conceptually different from the finite element method (FEM). FEM equations are usually partial differential equations (PDEs), while multibody equations are differential-algebraic equations (DAEs) or ordinary differential equations (ODEs). Also, FEM analyses are usually run in batch mode and are computationally expensive, whereas multibody simulations are associated with dynamic and real-time processes. Human-in-the-loop (HITL) and hardware-in-the-loop (HIL) applications are typical scenarios where multibody systems play an important role. Furthermore, multibody methods are very useful in the parameterization and design of mechanical systems due to their computational efficiency. In short, multibody dynamics constitute the computer science generalization of traditional methods (graphical and analytical) for the analysis of multi-rigid-body systems and mechanisms.

There are several commercial packages for the simulation of multibody dynamics in general and vehicle dynamics in particular. Some make simplifications in the vehicle topology or even linearize dynamic magnitudes (e.g. CarSim, Truck-

Sim, PC Crash[1]), while others consider the real geometry (e.g. numerical packages like Adams, Virtual.Lab Motion, RecurDyn, SIMPACK, SimMechanics, CarMaker, OptimumDynamics, and symbolic packages like MapleSim[2]). None of these are used in this Thesis, except for validation purposes, because an in-house software (MBS3D) has been developed and coded in MATLAB[3]/C/C++. This allows for the implementation of efficient multibody formulations providing great versatility, especially when novel techniques are to be implemented.

1.2.1 History

The theoretical foundations of multibody dynamics lie in the field of classical mechanics, namely, Newton-Euler's equations of motion for 3D solids (1687) augmented by D'Alembert's concept of constraints and reactions between solids (1743) and in the mathematical form developed by Lagrange (1788). The formal origin of multibody dynamics as a branch of physics can be set in 1977, at the "Dynamics of multibody systems" conference organized by K. Magnus in Berlin and sponsored by the International Union of Theoretical and Applied Mechanics (IUTAM). From that event to date, multibody dynamics has consolidated as a strong field of computational mechanics. In the rest of this section, the main milestones in the history of multibody systems are summarized.

The history of multibody dynamics has always been related to the computational advances of the time. The first practical problems to be solved, in the sixties, were open-loop systems like robots and satellites. To that end, relative coordinates between bodies (i.e., angles and distances allowed by the kinematic joints) were used because they were independent and required a small amount of memory. At that time, the first monographic book on multibody systems (Wittenburg, 1977) came out.

Later, in the seventies and eighties, vehicle applications with closed-loop topologies arose. These systems led to the use of reference point (absolute) coordinates, allowing the development of the first general-purpose (but less efficient) programs. Orlandea, Chace and Calahan, 1977, applied sparse matrix techniques and implicit ODE integrators to the equations of motion and constraint equations in reference point coordinates with Euler angles. This work allowed the development of the commercial package called Adams. At that time, a great deal of the research dealt with robotic systems, among which the solution of the inverse dynamics problem was crucial. Luh, Walker and Paul, 1980, presented an efficient method to solve it. The problem of forward dynamics was investi-

[1] CarSim, TruckSim and PC Crash are registered trademarks.

[2] Adams, Virtual.Lab Motion, RecurDyn, SIMPACK, SimMechanics, CarMaker, OptimumDynamics and MapleSim are registered trademarks.

[3] MATLAB is a registered trademark of The MathWorks, Inc.

gated further. One of the most efficient methods was the *method of the compound inertia*, due to Walker and Orin, 1982. Vereschagin, 1974, published the first fully recursive or O(N) method, and Featherstone, 1983, presented an improved formulation called *method of the articulated inertia*.

In the nineties and first years of the twenty-first century, research started facing new problems and applications like biomechanics, molecular dynamics, granular materials, co-simulation and multidisciplinary applications. Several text books on multibody dynamics were released: Nikravesh, 1988, Roberson and Schwertassek, 1988, Shabana, 1989, Haug, 1989, Huston, 1990 and Amirouche, 1992. Schiehlen, 1990, presented a catalogue with the characteristics of the existing multibody formulations and programs, followed by a paper about the state of the art of multibody dynamics (Schiehlen, 1997). Also, a relevant compendium of multibody formulations, with especial emphasis on natural coordinates[4] and real-time methods, was written by García de Jalón and Bayo, 1994.

Furthermore, during the first decade of the twenty-first century CPU clock-rate speeds stopped growing exponentially, and computer manufacturers started focusing on multiple processor architectures. On the other hand, graphic processing unit (GPU) manufacturers developed libraries with which the GPU processors could be used for carrying out custom mathematical calculations at very high speed. Thus, interest among the multibody community began to grow around parallel formulations. A few of the most important reviews were written by García de Jalón, 2007, Eberhard and Schiehlen, 2006, Laulusa and Bauchau, 2008, Bauchau and Laulusa, 2008.

In conclusion, multibody dynamics has proven to be a mature area of research with a strong application in industry, a growing presence in graduate and undergraduate university programs, and has led to the organization of frequent international conferences, e.g. ECCOMAS Multibody Dynamics, ASME International Conference on Multibody Systems, Nonlinear Dynamics, and Control (MSNDC), Asian Conference on Multibody Dynamics (ACMD) and Joint International Conference on Multibody System Dynamics (IMSD).

1.2.2 Mathematical principles

Several concepts have to be addressed before presenting multibody dynamics formulations. Among them, the concept of generalized coordinates and how these coordinates are used to analyze different kinds of system topologies. These ideas are then used to present the general theory of multibody dynamics.

[4] Cartesian coordinates of points and Cartesian components of unit vectors (see García de Jalón and Bayo, 1994, for further information).

Generalized coordinates Multibody systems are modeled using some kind of parameters to univocally define the position of the system, referred to as generalized coordinates. Usually, these coordinates are dependent, in the sense that there are more coordinates than the ones strictly necessary to fix the system position, i.e., there are more coordinates than degrees of freedom (DOFs). Basically, there are three types of generalized coordinates:

- Relative (or joint) coordinates. They are defined as the DOFs of the relative motions allowed by the joints, i.e., as coordinates between bodies. When used to model open-loop systems, joint coordinates are independent per se.
- Reference point (or absolute, or Cartesian) coordinates. They use the position of a point (normally the COG) to define the position of the body, and three angles or four Euler parameters to define the angular orientation.
- Natural (or fully Cartesian) coordinates. They consist of Cartesian coordinates of points and components of unit vectors that are used to define both the position and orientation of bodies, without the need of introducing additional angular quantities.

The number and type of coordinates affect the number, nature and sparsity of the motion differential equations, as well as the way displacements and forces are applied, thus being one of the most important modeling decisions.

Topology The second key aspect for the modeling of multibody systems is the topological nature of the mechanical system. The *topology* is the way bodies are connected by joints, and can be represented by a graph with bodies on the vertices and joints on the edges. The two basic topologies are open-loop and closed-loop chains. In the former, the motion of each joint is independent, while in the latter it may depend on the other joint motions of the loop. Examples of open-loop systems are certain satellites, serial manipulators and tree-structured multibody systems, while examples of closed-loop mechanisms include parallel manipulators and vehicle suspension systems. See Figure 1.4 for a symbolical representation of these systems. Open-loop systems are far easier to model and solve, whereas general closed-loop multibody systems require a higher level of elaboration and are numerically more sensitive.

In this Thesis, for the sake of generality and because vehicle suspension systems are inherently closed-loop mechanisms, all presented formulations are capable of solving open- and closed-loop systems.

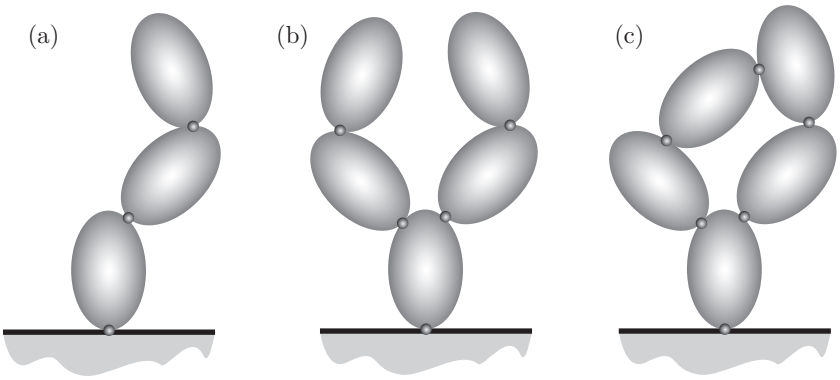


Figure 1.4: (a) Serial multibody system. (b) Tree-structured multibody system. (c) Closed-loop multibody system.

Lagrange's equations of the first kind In the descriptor form, using dependent Cartesian coordinates, the motion differential equations take the form:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{\Phi}_{\mathbf{q}}^{T}(\mathbf{q},t)\boldsymbol{\lambda} = \mathbf{F}(\mathbf{q},\dot{\mathbf{q}},t) \tag{1.1}$$

where $\mathbf{q} \in \mathbb{R}^n$ is the vector of Cartesian coordinates that defines the system position, $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ are its first and second order time derivatives, $\mathbf{M} \in \mathbb{R}^{n\times n}$ is the inertia or mass matrix, $\mathbf{F} \in \mathbb{R}^n$ is a vector that includes the external and velocity dependent inertia forces, $\mathbf{\Phi}_{\mathbf{q}} \in \mathbb{R}^{m\times n}$ is the Jacobian matrix of the kinematic constraint equations, and $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the vector of Lagrange multipliers. The position, velocity, and acceleration vectors in Eq. (1.1) must satisfy the corresponding constraint equations:

$$\mathbf{\Phi} \equiv \mathbf{\Phi}(\mathbf{q},t) = \mathbf{0} \tag{1.2}$$

$$\dot{\mathbf{\Phi}} \equiv \mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{\Phi}_{t} = \mathbf{0} \quad \rightarrow \quad \mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} = -\mathbf{\Phi}_{t} \equiv \mathbf{b} \tag{1.3}$$

$$\ddot{\mathbf{\Phi}} \equiv \mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{t} = \mathbf{0} \quad \rightarrow \quad \mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} = -\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} - \dot{\mathbf{\Phi}}_{t} \equiv \mathbf{c} \tag{1.4}$$

where constraint equations are assumed to be holonomic. Equations (1.1)-(1.4) constitute a system of index-3 DAEs. If only Eqs. (1.1) and (1.4) are considered, the following index-1 DAE system equivalent to an ODE system is obtained:

$$\begin{bmatrix} \mathbf{M} & \mathbf{\Phi}_{\mathbf{q}}^{T} \\ \mathbf{\Phi}_{\mathbf{q}} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ \mathbf{c} \end{Bmatrix} \tag{1.5}$$

The matrix in this system of linear equations is known as the *augmented matrix* (Negrut, Serban and Potra, 1997) or a matrix with *optimization structure* (Serban et al., 1997, von Schwerin, 1999). The system of differential equations (1.5) suffers from a numerical issue. As only the acceleration constraint equations have been imposed, the positions and velocities provided by the integrator do not fulfill their corresponding constraints and suffer from the *drift* phenomenon. Two popular solutions to this problem are the Baumgarte stabilization

method (Baumgarte, 1972) and the mass-orthogonal projections of position and velocity vectors (Bayo and Ledesma, 1996).

According to von Schwerin, 1999, Maggi's formulation can also be seen as an efficient way to solve the system of linear equations (1.5) and to, ultimately, integrate the dynamic equations of motion (1.1)−(1.4).

Formulations and implementation Due to the sensitivity of DAE systems to numerical errors, the variety of multibody formulations is quite large. A robust solution of Eqs. (1.1)−(1.4) requires a good choice of coordinates, a good formulation of the equations of motion and an effective integrator. The combination of these mathematical-computational strategies is referred to as multibody *formulation* or multibody *formalism*.

In general, multibody formulations can be classified into global and topological methods. The former address all systems in the same way, regardless of their topology, and are easily implemented into general-purpose programs. The latter focus and capitalize on the specific topology of the system, sometimes leading to very efficient implementations.

Finally, there are two basic families of implementations: numerical and symbolic. Numerical programs are valid for a large range of systems, and are usually implemented using matrix algebra and linear algebra libraries. Typically they are implemented in languages like C/C++, Fortran and MATLAB. Some examples of numerical software packages are MSC Adams, LMS Virtual.Lab Motion, SIMPACK and SimMechanics. On the other hand, symbolic approaches formulate the equations analytically and take advantage of the symbolic simplifications that can be carried out for each specific model. Thus, they save mathematical operations and are computationally more efficient. However, they are usually case-dependent and may have a complexity limit. Examples of symbolic software packages are MapleSim and Neweul. In this Thesis, all programs are numerically efficient (from the points of view of implementation and formulation) and have been numerically implemented using MATLAB and C/C++.

1.3 Optimization of multibody systems

Manual design of mechanical systems consists of improving the *quality* of a design by manually selecting and changing certain parameters of the system. Mankind has performed manual design for millenniums. Mathematical optimization is just an automatic way of solving design problems. Historically, optimization methods date back to the eighteenth century, when Gauss developed the steepest descent method. All mathematical optimization problems involve the improvement of some sort of objective function by changing the value of a set of design parameters algorithmically.

1.3.1 State of the art

In the context of multibody dynamics, there are different design problems that involve mathematical optimization, as well as different methods to solve them. Even though some of these problems are similar to the ones found in other disciplines like structural analysis or control theory, their peculiarities in the field of multibody dynamics are carefully listed here to avoid confusion. A review of the state of the art on sensitivity analysis and dynamic response optimization is also provided. Beware that only an overview is presented due to a lack of space. Sensitivity analysis and dynamic response optimization will be tackled in depth in the following chapters.

Sensitivity analysis It consists of analyzing and quantifying the effect of the parameters of the problem on the outputs. Sensitivity analyses are closely related to optimization procedures, as they provide relevant information about which parameters affect the design objectives to a greater extent. These are often performed prior to the optimization processes or in parallel.

The two main objectives of sensitivity analyses are the judgment of which design variables are relevant to the design objectives, and the computation of the gradient of each objective with respect to the design variables. Since the computation of sensitivities requires the differentiation of the equations of motion, it is a more complicated procedure than forward dynamics. Contrary to dynamic formulations, an efficient, accurate, completely general-purpose, and simple formulation for the analysis of sensitivities is, to date, not available.

Due to the fact that within the analysis of mechanical systems the sensitivity analyses stem from the equations of motion, there are as many sensitivity methods as forward dynamics formulations. Thus, the casuistry of methods might seem large. However, when classified by the underlying mathematical technique, there are two main methods: the adjoint variable method (AVM) and the direct differentiation method (DDM).

One of the first pieces of work on this area was published by Haug and Arora, 1978. In this work, the AVM was extended from control theory to the optimization of multibody systems. Later on, it was applied to more complex systems, but at the cost of a cumbersome formulation.

Shortly after, Sohoni and Haug, 1982, presented the sensitivity analysis of kinematically driven systems, which was then completed with the dynamic analysis by Haug, Wehage and Mani, 1984. That same year, the direct differentiation method (DDM) was presented by Krishnaswami and Bhatti, 1984, as a much simpler alternative to the AVM. However, the formulation still required the manual differentiation of complex terms.

Chang and Nikravesh, 1985, provided a good summary of DDM and AVM differences, although only through academic examples. Ashrafiuon and Mani,

1990, were the first to present a general-purpose method for the computation of sensitivities, based on symbolic computing.

A few years later, Pagalday, 1994, presented in his Thesis a general-purpose method for the symbolic and object oriented programming (OOP) optimization of multibody systems. That same year, Bestle, 1994, contributed an extensive description of sensitivity analysis and optimization of multibody systems.

At that point, direct differentiation approaches were very common for their generality and simplicity of implementation, and symbolic computation was widely used in the nineties to compute certain derivatives present in the sensitivity equations. However, there were still concerns and problems when applying sensitivity analysis methods, most of them coming from the practical implementation and differentiation of terms that appear both in the AVM and the DDM.

Bestle and Eberhard, 1992, developed an AVM implementation for holonomic and non-holonomic multibody systems, and applied it to an oscillator and a robot system, pointing out the superior accuracy of the AVM with respect to the numerical differentiation (ND) approach. Serban and Freeman, 1996, emphasized one of the main of characteristics of the DDM for the computation of sensitivities: the necessity of integrating both sensitivity and dynamic equations altogether. In 1996, Eberhard proposed the use of automatic differentiation (AD) to fill some of the shortages of symbolic differentiation in complex computer codes. AD is a computer-mathematical technique used to differentiate any type of function defined through computer code. By applying the chain rule of differentiation and augmenting the code with derivative terms that are propagated through the computational tree, machine-precision derivatives of complex computer programs can be computed semi-automatically.

Maly and Petzold, 1996, tackled the sensitivity analysis of general DAEs, and found that the use of ND had serious difficulties like the correct scaling of the problem and the accurate selection of the function perturbation. Liu, 1996, analyzed the effect of Baumgarte stabilization techniques on the integration of the sensitivity equations, specifically on the ones formulated using the AVM. A small example was solved for that purpose. Dias and Pereira, 1997, developed a DDM for rigid-flexible multibody systems based on the use of symbolic manipulators, and analyzed a slider-crank mechanism and a vehicle. Serban and Haug, 1998, demonstrated the speedup of manual (analytical) differentiation (MD) with respect to ND, using the DDM on an implicit, absolute-coordinate formulation, solving an 11-DOF vehicle case.

Li, Petzold and Zhu, 2000, studied the computational cost of the sensitivity analysis of DAEs computed with ADIFOR. Special emphasis was put on implementation and AD details. Wang, Haug and Pan, 2005, presented an exhaustive DDM for the computation of sensitivities, based on the coordinate partitioning

method and the state-space approach, which were then applied to the implicit integration of the motion differential equations and the sensitivity equations. Schaffer, in 2005, developed in his Thesis the adjoint piecewise method, which overcame some of the disadvantages of the AVM by using parallelization techniques. The method combined advantages both from the AVM and the DDM. Chatillon et al., 2006, tackled the problem of vehicle sensitivities as a way of quantifying the mathematical uncertainty of the model parameters. Even though they also performed interesting multi-objective optimization, only approximated vehicle models were used. Ding, Pan and Chen, 2007, presented a method for the second-order sensitivity analysis of DAEs based on the AVM, and supported it with a slider-crank mechanism example. Brüls and Eberhard, 2008, investigated the sensitivity analysis of systems with finite rotations by using the DDM and geometric integrators, analyzing an academic example.

Pi, Zhang and Chen, 2012, provided a good literature review and developed a manual (analytical) sensitivity analysis formulation for flexible multibody dynamics, based on the Absolute Nodal Coordinate Formulation (ANCF) and the direct differentiation approach. The results presented for small size systems were good, but the efficiency was not fully assessed. Sonneville and Brüls, 2013, developed adjoint-variable and direct-differentiation methods for multibody systems formulated on a Lie group, pointing out their advantages and disadvantages on two medium-size response optimization problems. Even though their approach showed great potential, the results seemed rather inconclusive. Banerjee and McPhee, in 2013, presented an overview of the possibilities of symbolic computing for the calculation of sensitivities, which they backed by simulating a slider-crank mechanism.

Summing up, in the last 30 years there have been numerous attempts to solve the sensitivity analysis problem in a simple and general way. However, most methods are still not general-purpose and are cumbersome for real-life models like vehicles. In Chapters 4 and 5 this problem is tackled from a novel angle in the context of general-purpose multibody systems.

Geometrical and kinematic optimization Geometrical optimization is also referred to as *mechanism synthesis*. It deals with the optimization of geometric quantities, normally related to path-planning design of robots. The objective function merely depends on the geometry of the system, and neither the inertia properties of the bodies nor the forces are considered.

In the literature, many case-dependent methods for mechanism synthesis are found, but only a few of them are general methods. Synthesis methods have a very long trajectory in the field of multibody systems. Three different types of problems are usually enumerated: path-following, function generation and body guidance. Path-following problems modify the system geometry so that, for in-

stance, the position of the robot end-effector follows a predefined trajectory. More complex applications can also be found in the literature (see Avilés, Ajuria and García de Jalón, 1985, and Gómez-Cristóbal, 2003). Also, a good overview of engineering design and mechanism synthesis problems and solutions is given in Erdman, 1995.

In these types of problems, a robust way of dealing with assembly constraints, singular positions and lock-up positions is needed. Otherwise, the Jacobian of the constraint equations might become ill-conditioned with a particular choice of geometrical dimensions and the optimization might yield inaccurate results. To illustrate the topic with a vehicle dynamics example, see De-Juan, Sancibrian and Viadero, 2012. In their work, the design of a vehicle suspension system is carried out, with the objective of arriving at an optimal motion.

Another variant of geometrical optimization is *kinematic optimization*. It is sometimes called *kineto-static optimization*. In this approach, certain kinematic variables (not only the geometry) are part of the objective function and can be constrained. As in geometrical optimization, the equations of motion need not be integrated. In the case of spatial parallel manipulators, the objective function is often connected to the relationship between the actuated velocities and the platform velocities. This relationship can be evaluated in terms of the conditioning of the Jacobian matrix, leading to performance coefficients such as: isotropy, dexterity, manipulability, condition index, local mobility index, error amplification and stiffness properties (Collard, 2007).

Dynamic response optimization It could also be called *dynamic optimization*, but since the concept of dynamic optimization is used differently within optimal control, the first name is normally used in the context of multibody dynamics. Response optimization, as the name suggests, studies how the dynamic behavior of dynamical systems can be improved. Mathematically, it implies the analysis of transient dynamic quantities like displacements, velocities, accelerations and forces, and how they are influenced by the geometry, the external forces, the inertia parameters, etc. It is therefore the most comprehensive problem, because all types of design parameters may be optimized. Also, the integration of the motion differential equations implies an additional complexity and has to be handled with specific methods.

Contrary to mechanism synthesis, a topic on which an abundant literature can be found, dynamic response optimization of multibody systems has been studied to a lesser extent. Response optimization is a recipe that gathers several sensitive ingredients, namely: highly nonlinear kinematic and optimization constraint equations; the time integration of motion differential equations that can sometimes be numerically stiff; a numerically expensive sensitivity analysis; and the

possible presence of multiple objective functions, which makes the problem even harder to solve.

Paeng and Arora, 1989, proposed a multiplier method for the dynamic response optimization of mechanical systems, although they only provided academic examples. This was one of the first pieces of work in the literature. A few years later, Besselink and Van Asperen, 1994, applied numerical optimization of the comfort to a linear structural model of a tractor vehicle, using numerical sensitivities. They pointed out the usefulness of the sensitivity analysis at the optimum solution point. Pagalday, 1994, and Pagalday and Avello, 1997, presented one of the first general-purpose implementations for the optimization of multibody systems, including an accurate computation of sensitivities, as well as a good overview of the context of dynamic response optimization of multibody systems. Specifically, they proposed a symbolic-algorithmic tool for the differentiation of mathematical terms, which, from the point of view of the author of this Thesis, constitutes the closest precursor of AD for sensitivity computation in multibody systems. Two relatively large examples were solved with the proposed method.

Etman, Van Campen and Schoofs, 1998, used local approximation techniques and the SQP algorithm to optimize three academic examples. As tools for the DDM and the AVM, symbolic, automatic and hybrid numerical-analytical derivatives were recommended. Also, Baumal, McPhee and Calamai, 1998, used a simplified multibody model with active suspension to perform global optimization with a genetic algorithm (GA), obtaining a 4% improvement with respect to the gradient-based method. The same year, Kim and Choi, 1998, presented an Augmented-Lagrangian-based method for the solution of min-max optimization problems in mechanical systems, but only academic examples were studied.

Eberhard, Schiehlen and Bestle, 1999, stressed the advantages of stochastic methods over local methods, especially to find global minima and to show the entire Edgeworth-Pareto optimal set. They applied these methods to a simplified multibody car, and proposed the use of hybrid local-global methods, although stating that the switching strategy was still a challenging issue. Hegazy, Rahnejat and Hussain, 2000, optimized the behavior of a car while undergoing a double lane-change maneuver, giving useful insight about the relevant dynamic variables. Eriksson and Friberg, 2000, improved the comfort response of a city bus by 7%, through use of an FEM chassis and the SQP optimization method. Serban and Freeman, 2001, presented a nonlinear least-squares optimization approach for the parameter identification of multibody systems, by using the DDM to compute sensitivities. The formulation was applied to an academic example and a realistic multibody vehicle.

Naudé and Snyman, 2003, optimized the comfort behavior of a simplified 2D model, obtaining large improvements in a short computation time, using a leap-frog optimization method. Haghiac, Haque and Fadel, 2004, applied stochastic optimization methods to the vehicle handling of an approximated vehicle model. They considered a large number of design parameters and performance indices, focusing on the quality of the solution rather than on the method efficiency. Three maneuvers were considered: J-turn, single sinusoidal and double lane-change. Andersson and Eriksson, 2004, performed a comprehensive ride and handling optimization of an intercity bus modeled using Adams. They carried out an scalarization of several objective functions into a single index and used the SQP algorithm.

Gonçalves and Ambrósio, 2005, optimized the suspension of a sports car with respect to handling, with ride comfort as a constraint. The multibody model considered the flexibility of the chassis, and was described in depth. Kübler, Henninger and Eberhard, 2005, optimized an hexapod machine by using the SQP algorithm together with finite-difference and AD gradients, although with little detail provided of the latter.

Polach and Hajžman, 2008, carried out a semi-automatic design of shock-absorbers, based on the dynamic response of an intercity bus when riding over a bump. The approach followed a quasi-parameter-identification technique. Thoresson et al., 2009, published a two-part article that started with a detailed state of the art of vehicle suspension optimization. Then, they used the Dynamic-Q method with finite-difference sensitivities and a scalarized objective function to optimize a car modeled in Adams with respect to ride and comfort. However, they had to use approximated models in order to reduce numerical noise and speed up the simulations. Özcan, Sönmez and Güvenç, 2013, provided a thorough and clear review of dynamic response optimization approaches for vehicle models, as well as handling and comfort metrics. They also applied suspension design to lumped quarter-car, lumped half-car and full-vehicle suspensions. However, they reported small improvements because the initial configuration was very close to the optimal one.

Topological optimization It is very well known for its applications on structural analysis (for a good overview, see Bendsoe and Sigmund, 2003). Topological optimization of multibody systems tries to find the best structural configuration for a specific objective function. To that end, the topology of the system (i.e., the connectivity of the bodies) is defined in terms of the design parameters. The algorithm then evaluates the designs based on the objective function and the constraints, and iterates until the design is optimum. The reticulated nature of multibody systems, their inherent nonlinearities and the variety of topologies make it difficult to derive general formulations. Nevertheless, approaches such as the ones from Ceccarelli, 1993, McGarva, 1994, Kawamoto,

2004, Kawamoto, 2005, and Stolpe and Kawamoto, 2005, yield interesting results. Recent techniques also include graph and enumeration methods.

Parameter identification Consists of fitting a given, fixed mathematical model to experimentally measured data, by adjusting the model parameters. This is a key problem in many multibody system applications, where some of the parameters cannot be calculated easily (e.g. the inertia of a vehicle chassis or the engine weight). One of the biggest challenges in parameter identification is the detection of the dependencies between parameters, as well as the linearity of the model with respect to the model parameters.

Gauthier and Khalil, 1990, enumerated a set of rules, based on the system topology, to group dependent parameters. Fisette, Raucent and Samin, 1996, proposed a recursive method to obtain the minimal formulation. With the same purpose, Moore, Kövecses and Piedboeuf, 2003, presented a method based on symbolic tools. Chen and Beale, 2003 found the minimum set of parameters by applying a singular value decomposition (SVD) on the simulation results.

Numerical optimization has been widely used to find the optimal value of the unknown parameters of the model (see, for instance, Serban and Freeman, 2001). The objective function is the difference between the model response and the experimental response, while the design parameters are the unknown model characteristics. Furthermore, by computing the sensitivities of the model, one can assess the dependency of the outputs on the model parameters.

Other techniques based on the Kalman filter (Kalman, 1960) are recently gaining acceptance. One of the main advantages of this technique is that it is less sensitive to the noise in the measured data. Also, it is said to be efficient, accurate and easy to implement, but it does not converge if the mathematical model is not accurate enough.

1.3.2 Mathematical principles

Before tackling the optimization of the dynamic behavior of complex mechanical systems, a few general optimization concepts must be defined. The following definitions are valid for any standard optimization problem, and will be used extensively throughout this Thesis.

Design parameters The variables of the system (and therefore of the mathematical model) that can be modified by the user (and ultimately by the optimization method) to improve the quality of the system. Design variables are a subset of all variables that affect the performance of the system. They are chosen by the designer, and are normally grouped in the following vector

$$\mathbf{b} \equiv \{b_1, b_2, ..., b_{n_{\text{dp}}}\}^T \tag{1.6}$$

where n_{dp} is the number of design parameters. There are continuous design variables (such as the length of a beam) and discrete design variables (such as the number of wheels on a railway car). Only parameters that affect the objective function can be considered as design parameters. As the magnitude (or even the existence) of the relationship between the variables and the objective function is often not evident, a preliminary step called *sensitivity analysis* is carried out before the optimization process. Chapter 5 is devoted to this issue.

Within response optimization procedures, the first type of design parameters are *auxiliary parameters*. Basically, this type of parameter encompasses variables that are constant over the whole simulation and are usually linked to the kinematic guidance of coordinates or with external forces. Examples of these parameters are stiffness coefficients of springs and damping coefficients of dampers.

The second family of design parameter are *inertia parameters*. These parameters define the inertia properties of the bodies, namely the position of the COG, the mass, and the elements of the inertia tensor. If the ten inertia parameters of each solid are not independent, these relationships should be explicitly reflected on the model.

The last type of design parameter are *geometric parameters*. The local geometry of the bodies, that is, the position and orientation of the joints with respect to the local reference frame, can also be considered as design variables. These include, for instance, unit vectors defining revolute joints, points defining spherical joints, etc. They are the parameters considered by mechanism synthesis, while not as common in response optimization problems. The reason is that the geometry of the system affects the kinematic constraints, the mass matrices and the force vectors, and the optimization formalism has to be robust enough to be able to handle them.

Objective function It is the mathematical quantification of the quality of the system. The objective function is a function of the design parameters:

$$\Psi_0 = \Psi_0(\mathbf{b}) \tag{1.7}$$

It is sometimes called *cost function* because in many optimization problems the quality is measured as an economic cost. Accordingly, most of the optimization problems try to reduce the cost function and are formulated as minimization procedures. Examples of objective functions in engineering are the system weight, cost, noise, etc. Reducing the objective function thus improves the quality of the system. The goal of optimization methods is to find an optimum set of parameters $\mathbf{b}^*$ such that the value of the objective function is minimum:

$$\Psi_0(\mathbf{b}^*) \leq \Psi_0(\mathbf{b}), \quad \forall \mathbf{b} \in \Omega_{\mathbf{b}} \tag{1.8}$$

where $\Omega_{\mathbf{b}}$ is the design variable space made up of all possible values of $\mathbf{b}$. Sometimes it is not possible to define a single objective function, and several objective functions are required. This problem is tackled in Chapter 6.

In the field of dynamic response optimization, the dependency of the objective function on the design parameters is often both explicit and implicit. The dynamic response depends explicitly on the dynamic state of the system (time, positions, velocities and accelerations), which in turn might also depend on the design parameters. Therefore, the general form of the objective function in the context of multibody systems would be:

$$\Psi_0 = \Psi_0\left(t, \mathbf{z}(t,\mathbf{b}), \dot{\mathbf{z}}(t,\mathbf{b}), \ddot{\mathbf{z}}(t,\mathbf{b}), \mathbf{b}\right) \tag{1.9}$$

where $\mathbf{z}$, $\dot{\mathbf{z}}$, and $\ddot{\mathbf{z}}$ are the generalized positions, velocities and accelerations, respectively. These dependencies will sometimes not be written explicitly in this Thesis for the sake of brevity, but will be taken into account later on as far as differentiation is concerned.

The form of the objective function largely depends on the specific scientific field and application. In the case of multibody dynamics, there are a few typical objective functions which cover most of the dynamic response optimization problems. The first and most basic objective function is the one that only depends on the design, which was shown in Eq. (1.7). Another typical objective function is the *point objective function*:

$$\Psi_0 = \max_t f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) \tag{1.10}$$

This type of objective function represents the maximum value of a dynamic quantity over time. Examples of dynamic magnitudes that the designer might want to minimize are joint reactions and accelerations.

Another typical expression of the objective function is the *integral objective function*, which is an integral over the simulation time (from the initial time t_i to the final time t_f) of a function depending on the dynamic states and the design variables:

$$\Psi_0 = \int_{t_i}^{t_f} f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b})\, dt \tag{1.11}$$

These objective functions are often used to minimize the mean value of quantities such as accelerations or energy over time. The last type of objective functions within response optimization are *pointwise objective functions*, where the response of the system has to match predefined values at specific time points:

$$\Psi_0\left(t_k, \mathbf{z}_k, \dot{\mathbf{z}}_k, \ddot{\mathbf{z}}_k, \mathbf{b}\right) = 0 \tag{1.12}$$

This type of objective function, while not used in this Thesis, can be applied, for instance, in control approaches where the response trajectory of the system has

to match a predefined path. It can also be useful when time-dependent objective functions are replaced by pointwise constraints, as will be explained further on.

Optimization constraints Mathematical variables can take any value, but real variables are often limited by their own nature and/or by design constraints. Examples of nature and design limitations are the fact that masses can only be positive or a maximum acceleration imposed by the designer. Constraintless optimization problems normally lead to impractical results, and are only posed in theoretical approaches.

Optimization constraints are mathematical expressions of the design variables that limit their values or establish explicit relationships between them. The set of design parameters that satisfy the constraint equations constitute the *feasible set* of design parameters or *design parameters space*. The simplest expression of optimization constraints is:

$$\Psi_i = \Psi_i(\mathbf{b}) \tag{1.13}$$

which can then be grouped in vector

$$\boldsymbol{\Psi} \equiv \{\Psi_1, \Psi_2, ..., \Psi_{n_{\text{oc}}}\}^T \tag{1.14}$$

where n_{oc} is the number of optimization constraints and Ψ_i is the i-th optimization constraint. Note that the concept of constraints in the context of optimization methods is similar but not equal to the kinematic constraints previously presented in the introduction to multibody system dynamics.

Following the same approach as with the objective functions, dynamic response optimization constraints can be expressed in terms of their dependencies as:

$$\Psi_i = \Psi_i\left(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}\right) \tag{1.15}$$

Constraints can be classified as equality, inequality, box, point and integral constraints. *Inequality constraints* are of the following form:

$$\Psi_i \equiv f\left(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}\right) \leq 0 \tag{1.16}$$

This type of constraint divides the space into two half-spaces and only allows one half. More restrictive and less typical type of constraints are *equality constraints*, which adopt the following form:

$$\Psi_i \equiv f\left(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}\right) = 0 \tag{1.17}$$

which can be rewritten as two inequality constraints:

$$\Psi_1 \equiv f\left(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}\right) \leq 0 \tag{1.18}$$

$$\Psi_2 \equiv -f\left(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}\right) \leq 0 \tag{1.19}$$

A very common type of constraint is the one that sets upper and lower bounds to design variables. They are called *box constraints*, and are present in almost

all engineering optimization problems to protect the physical nature of the variables and enforce material limits. They are formulated as:

$$b_i^l \leq b_i \leq b_i^u \tag{1.20}$$

where b_i^l and b_i^u are the lower and upper bounds, respectively. As was done with equality constraints, they can be rewritten as two inequality constraints:

$$\Psi_1 \equiv b_i - b_i^u \leq 0 \tag{1.21}$$

$$\Psi_2 \equiv b_i^l - b_i \leq 0 \tag{1.22}$$

Box constraints are sometimes treated in a special way by optimization algorithms in order to increase efficiency.

Summing up, the three types of optimization constraints presented are particular cases of two general types: point and integral constraints. Analogously to the case of objective functions, *point constraints*, *integral constraints* and *pointwise constraints* take the following form:

$$\Psi_i \equiv \max_t f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) \leq 0 \tag{1.23}$$

$$\Psi_i \equiv \int_{t_i}^{t_f} f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b})\, dt \leq 0 \tag{1.24}$$

$$\Psi_i \equiv \Psi_i\left(t_k, \mathbf{z}_k, \dot{\mathbf{z}}_k, \ddot{\mathbf{z}}_k, \mathbf{b}\right) = 0 \tag{1.25}$$

These constraints allow restraining the maximum value of a function of the dynamic variables over time, the mean value of certain quantities over time, and the value of magnitudes at certain time points. These types of constraints are especially suited to remove the time dependency from the optimization constraints, as they are a scalar measure of the whole integration time. However, maximum-type functions are not differentiable and can lead to problems with the computation of gradients in the optimization process.

Alternative expressions Since objective functions and constraints often have similar mathematical expressions and optimization methods treat them similarly, it is sometimes useful to combine the expressions into generic equations. This simplifies the theoretical development as well as the implementation. Firstly, point objective functions like the one in Eq. (1.10) can be expressed in terms of constraints using the following strategy: a new design variable $b^{\max}$ is introduced into the problem and the original objective function, Ψ_0, is replaced by a new objective function and a constraint in the form:

$$\Psi_0 = b^{\max} \tag{1.26}$$

$$\Psi_1 \equiv \max_t f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) - b^{\max} \leq 0 \tag{1.27}$$

That way, the reduction of $b^{\max}$ enforces the original objective function indirectly by decreasing the limit of the constraint equation. This strategy can be useful if 'maximum' functions are not handled properly by the optimization routine. Also, point constraints like the one in Eq. (1.23) can be replaced by the following integral-type expression:

$$\Psi_i \equiv \int_{t_i}^{t_f} \left(f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) + \left| f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) \right| \right) dt = 0 \tag{1.28}$$

which can then be replaced by two inequality constraints as explained before.

Whether point equations are replaced by integral functions or not, objective functions and constraints can be expressed using the following generic equation:

$$\Psi_i \equiv \max_t f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) + \int_{t_i}^{t_f} g(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b})\, dt \leq 0 \tag{1.29}$$

which is valid for objective and constraint functions and summarizes the way an optimization problem is written mathematically.

As Etman, 1997, points out, sometimes authors carry out function transformations to avoid the dependency of the objective and constraint functions with time and to avoid discontinuities in specific functions. When the transformation consists of discretizing the constraints into a set of *pointwise constraints*, the number of constraints increases considerably and there is a risk of non-enforcement of constraints between the time points. When time dependent constraints are replaced by integral type constraints, these transformations might sometimes alter the nature of the objective and constraint equations. These transformations are not used here because the optimization algorithms employed could handle time dependent constraints. Should the objective and constraint functions undergo discontinuities or undesired effects, alternative formulations of these equations must be developed.

In problems where the number of constraints is very large and thus the computational cost is significant, techniques like *constraint screening* or *constraint deletion* should be applied. The first one consists of finding out which constraints are less numerically relevant for the minimization of the specific objectives, while the second consists of deleting those constraints.

In the following chapters, especially in Chapter 6, further details are given about the implementation of optimization methods and the results obtained.

1.3.3 Automatic differentiation

Automatic or *algorithmic differentiation* (AD) is a computational technique that augments a computer code, no matter how complex, with the derivatives of the variables with respect to the independent variables (or *inputs*). This enables an automatic computation of first- and higher-order derivatives like gradients, Ja-

cobians, Hessians, etc. AD has been applied to all kinds of computer codes, including Physics, Chemistry, Medicine, Biology and Engineering, among others. In the field of multibody dynamics, a few pieces of work have been presented, but there are still uncertainties about its efficiency and applicability to real problems. An important part of this Thesis (Chapters 4 and 5) deals with the AD of multibody formulations. Next, a general overview of relevant past and recent AD developments is given.

In the early nineties, Juedes, 1991, analyzed the available AD tools and their main characteristics. Barthelemy and Hall, 1992, explained with very clear examples, expressions and diagrams, the way AD worked, drawing performance conclusions from AD tools of the time. Griewank and Reese, 1992, gave useful insight about the efficient computation of Jacobian matrices. Specifically, they explained how the application of the Markowitz rule is equivalent to the systematic application of the chain rule on a computational graph.

Bischof et al., 1996, stressed the fact that efficiency conclusions about codes differentiated with ADIFOR cannot be easily drawn, as different problem sizes and implementations might show very different performances. Bischof, 1996, used ADIFOR to run a sensitivity analysis on a multibody model of an Iltis vehicle, although little detail was given about the mechanical model and the multibody formulation. Eberhard, 1996, analyzed, in the context of multibody systems, the AVM and the reverse mode of AD, and found that both methods have interesting similarities, even at the code level. Yet, the AVM showed higher efficiency, while AD showed higher generality and flexibility.

Eberhard and Bischof, 1999, studied the AD of time integration algorithms, specifically for the sensitivity analysis of mechanical systems. In their work, they focused on the diverse differentiation and integration schemes and their corresponding numerical errors. As an example system, they simulated a 5-DOF robot by using ADIFOR. Furthermore, Enciu, Gerbaud and Wurtz, 2010, based on the work by Eberhard, considered the sensitivity analysis of variable-length simulations so as to perform numerical optimization. However, the number of DOFs was still small.

Stadler and Eberhard, 2001, used ADIFOR to compute the sensitivity equations of a manipulator end-effector, by differentiating the motion differential equations of the multibody system. Bras and Azevedo, 2001, presented a simple but meaningful example of the optimization of a small mechanical system using AD techniques, including an explanation of AD based on Rall numbers (Rall, 1981) and the techniques of variable scaling and constraint normalization. Naumann and Walther, 2003, provided a useful review of AD tools, including implementation details. Also, an explanation was given about how the Curtis-Powell-Reid seeding for the exploitation of Jacobian sparsity works.

Eberhard, Schiehlen and Sierts, 2007, used AD to run a sensitivity analysis of the inertia parameters of a car. However, little detail was given about the differentiation tool and algorithm. Bischof, Hovland and Norris, 2008, presented an excellent explanation of AD principles, together with instructive examples and a taxonomy of AD tools. Special emphasis was given to ADIC2, and the authors predicted the advent of hybrid operator-overloading and source-transformation tools. Siskind and Pearlmutter, 2008, paid attention to the manual (non-automatic) work involved in the implementation of AD tools. Specifically, they tried ADIFOR, Tapenade, ADIC and Fadbad++ on the same code, including nested differentiation.

From this brief literature summary, it seems that AD can play an important role in the semi-automatic derivation of the equations of motion and of the sensitivity equations of mechanical systems. However, its great potential has somehow been slowed down by the rather lengthy development of AD tools, which has prevented its application to real-life, general-purpose, complex formulations, at least in the multibody dynamics field. Two state-of-the-art applications of AD to multibody systems will be presented in Chapters 4 and 5.

1.4 Motivation and objectives

The optimization of the dynamic response of vehicles produces two very interesting results, namely sustainability and safety. Optimization improves the performance of vehicles and improves their handling and comfort characteristics, which in the long term reduces the number of accidents and improves driving experience. Moreover, optimization leads to efficiency, and efficiency leads to sustainability, which, for obvious reasons, are seen as major research objectives these days. However, optimization requires the development of realistic vehicle models and state-of-the-art sensitivity analysis techniques. To date, the optimization of large and complex multibody systems is still an open topic, as has been partially shown in the Introduction.

First of all, a few vehicle dynamics concepts have been presented, and the use of the multibody systems approach for the simulation of vehicles has been justified. Efficient multibody formulations are still a very active research topic too. Robust and efficient formulations for the simulation of multibody systems are essential for many applications like real-time dynamics, efficient control, parameter identification, and, of course, dynamic response optimization. It has been made clear, as well, that the transition from academic 2D models to real-life 3D models increases the simulation times in a prohibitive manner, thus preventing the use of certain formulation for real applications.

Second, a summary of the most relevant ideas on the optimization of multibody systems has been presented, starting from general optimization concepts and

ending with the most typical optimization problems and methods. A historical overview has also been described, which might help understand the most recent advances in sensitivity analysis and response optimization. Although many interesting developments have been developed in the field of dynamic response optimization since Haug and Arora, 1978, presented their groundbreaking work, many questions and challenges still remain unanswered.

Third, from the author's point of view, the simulation and optimization of multibody systems is only as good as its applicability to real-life problems. This demands something that previous work in the literature lacks for the most part: a general-purpose nature and the simulation of large, real-life examples that prove it. Multibody formulations should nowadays be mature enough so as to handle complex systems without too much trouble.

Fourth, and last, recent Doctoral Theses on this matter outlined the necessity of going further into the aforementioned topics, among others: Pagalday, 1994, Etman, 1997, Gonçalves, 2002, Schaffer, 2005, Vidal, 2006, Thoresson, 2007, Pasquotti, 2008, Hidalgo, 2013, and Banerjee, 2013. During the development of this Thesis, these dissertations have served as basic references, as they represent valuable sources on multibody dynamics, vehicle dynamics, optimization of multibody systems, sensitivity analysis of multibody systems and mechanism synthesis. Also, an analysis of the related bibliography was carried out (see the References chapter), revealing that there is still room for improvement and quality research on these areas. Hopefully, this Thesis will also contribute to the advance of the dynamic response optimization of multibody systems.

1.5 Objectives and structure

The remaining structure of the Thesis is detailed next, in accordance with the Thesis objectives, which are fourfold:

– First, implement an efficient formulation for the dynamic simulation of multibody systems, including a 3D graphics viewer. Develop a realistic coach model, including contact, driving and component forces, as well as virtual test setups, such that real vehicle maneuvers and dynamic responses can be accurately analyzed both from handling and ride comfort points of view. Chapters 2 and 3 are devoted to these topics.

- Second, carry out a benchmark of state-of-the-art tools for the automatic differentiation of C/C++ computer codes, including the simulation of multi-body systems with different sizes and configurations, in order to assess the performance of such tools in this particular context. Then, apply the most efficient automatic differentiation approach to the development of a general-purpose, hybrid, direct-automatic method for the computation of state and design sensitivities. These tasks are undertaken in Chapters 4 and 5.
- Third, apply the dynamic formulation and sensitivity analysis capabilities to the dynamic response optimization of the coach, both from handling and ride comfort perspectives. Appropriate local and global optimization methods will be used, together with meaningful maneuvers, objective functions and optimization constraints. This topic is covered in Chapter 6.
- Fourth and last, assess the advantages of the presented methods and set out possible approaches for future development (Chapter 7).

Chapter 2

Efficient multibody dynamics formulation

The optimization of multibody systems involves the solution of several problems, including the kinematic, the dynamic and the optimization ones, among others. Within each of them, several subproblems can be differentiated. The forward (dynamic) simulation of multibody systems is one of them, and is tackled here. Forward dynamics constitute a crucial step of the design process, providing the basis of the dynamic response optimization carried out later.

According to a recent paper from Laulusa and Bauchau, 2008, Maggi's formulation is a simple and stable way to solve the dynamic equations of constrained multibody systems. Among the difficulties of Maggi's formulation, Laulusa and Bauchau quoted the need for an appropriate choice (and change, when necessary) of independent coordinates, as well as the high cost of computing and updating the basis of the tangent null space of constraint equations. Here, an improved double-step method, which implements the matrix transformations of Maggi's formulation in an efficient way, is described. Some of the ideas presented here are also described in García de Jalón et al., 2005, García de Jalón, Hidalgo and Callejo, 2009, and García de Jalón, Callejo and Hidalgo, 2012.

2.1 Null-space method

Von Schwerin, 1999, describes an interesting way of solving the systems of equations in Eq. (1.5): the Null Space Method (NSM). In what follows, the notation of Eq. (1.5) will be used, taking into account that if matrix $\mathbf{\Phi_q}$ is not of full rank, an appropriate LU factorization should be performed, as indicated in García de Jalón, Callejo and Hidalgo, 2012.

Following the NSM, Lagrange multipliers can be eliminated from expression (1.5). This is traditionally done by calculating a basis for the kernel or null space of the Jacobian matrix $\mathbf{\Phi_q}$ (or $\mathbf{U}$, in the equivalent case). This basis can

be determined in several ways. Perhaps, the simplest one is the coordinate partitioning method (Wehage and Haug, 1982, Serna, Avilés and García de Jalón, 1982), which divides the coordinates $\mathbf{q}$ (and the columns of $\mathbf{\Phi}_\mathbf{q}$) into dependent and independent ones:

$$\begin{bmatrix}\mathbf{\Phi}_\mathbf{q}^d & \mathbf{\Phi}_\mathbf{q}^i\end{bmatrix}\begin{Bmatrix}\ddot{\mathbf{q}}^d \\ \ddot{\mathbf{q}}^i\end{Bmatrix} = \{\mathbf{c}\} \tag{2.1}$$

in such a way that matrix $\mathbf{\Phi}_\mathbf{q}^d$ is invertible or, at least, of full column rank (in order to have a left inverse). If $\ddot{\mathbf{z}}$ stands for the independent accelerations and $f = n - r$ for the number of DOFs, system (2.1) can be expressed as:

$$\begin{bmatrix}\mathbf{\Phi}_\mathbf{q}^d & \mathbf{\Phi}_\mathbf{q}^i \\ \mathbf{0}_{f\times m} & \mathbf{I}_{f\times f}\end{bmatrix}\begin{Bmatrix}\ddot{\mathbf{q}}^d \\ \ddot{\mathbf{q}}^i\end{Bmatrix} = \begin{Bmatrix}\mathbf{c} \\ \ddot{\mathbf{z}}\end{Bmatrix} \tag{2.2}$$

Assuming that the system matrix in (2.2) is invertible, if $\begin{bmatrix}\mathbf{S} & \mathbf{R}\end{bmatrix}$ is its inverse matrix, it can be written that:

$$\begin{bmatrix}\mathbf{\Phi}_\mathbf{q}^d & \mathbf{\Phi}_\mathbf{q}^i \\ \mathbf{0}_{f\times m} & \mathbf{I}_f\end{bmatrix}\begin{bmatrix}\mathbf{S}^d & \mathbf{R}^d \\ \mathbf{S}^i & \mathbf{R}^i\end{bmatrix} = \begin{bmatrix}\mathbf{I}_m & \mathbf{0}_{m\times f} \\ \mathbf{0}_{f\times m} & \mathbf{I}_f\end{bmatrix} \tag{2.3}$$

from which matrices $\mathbf{S}$ and $\mathbf{R}$ are found in the form:

$$\mathbf{S} = \begin{bmatrix}\mathbf{S}^d \\ \mathbf{0}_{f\times m}\end{bmatrix} = \begin{bmatrix}\left(\mathbf{\Phi}_\mathbf{q}^d\right)^{-1} \\ \mathbf{0}_{f\times m}\end{bmatrix} \tag{2.4}$$

$$\mathbf{R} = \begin{bmatrix}\mathbf{R}^d \\ \mathbf{I}_f\end{bmatrix} = \begin{bmatrix}-\left(\mathbf{\Phi}_\mathbf{q}^d\right)^{-1}\mathbf{\Phi}_\mathbf{q}^i \\ \mathbf{I}_f\end{bmatrix} \tag{2.5}$$

Note that the columns of matrix $\mathbf{R}$ are a basis for $\ker(\mathbf{\Phi}_\mathbf{q})$, whereas matrix $\mathbf{S}$ is a right inverse of $\mathbf{\Phi}_\mathbf{q}$. It is then verified that:

$$\mathbf{\Phi}_\mathbf{q}\mathbf{R} = \mathbf{0}_{m\times f} \tag{2.6}$$

$$\mathbf{\Phi}_\mathbf{q}\mathbf{S} = \mathbf{I}_m \tag{2.7}$$

The null-space formulation eliminates the Lagrange multipliers of system (1.5) by pre-multiplying the upper part of the aforesaid equation by $\mathbf{R}^T$ and considering Eq. (2.6). The following system is obtained:

$$\begin{bmatrix}\mathbf{R}^T\mathbf{M} \\ \mathbf{\Phi}_\mathbf{q}\end{bmatrix}\{\ddot{\mathbf{q}}\} = \begin{Bmatrix}\mathbf{R}^T\mathbf{F} \\ \mathbf{c}\end{Bmatrix} \tag{2.8}$$

The necessary and sufficient condition for system (2.8) to be compatible and have a unique solution is:

$$\operatorname{rank}\left(\begin{bmatrix}\mathbf{M}\\ \mathbf{\Phi_q}\end{bmatrix}\right) = n$$

Lagrange multipliers can be found by pre-multiplying Eq. (1.1) by $\mathbf{S}^T$ and considering Eq. (2.7):

$$\boldsymbol{\lambda} = \mathbf{S}^T\mathbf{F} - \mathbf{S}^T\mathbf{M}\ddot{\mathbf{q}} \tag{2.9}$$

From the point of view of efficiency, the null-space method is not very different from index-1 Lagrange's equations, and it has the same instability problems in the numerical integration process because it does not consider the position and velocity constraints (1.2)-(1.3).

Instead, Maggi's method poses the differential equations only in terms of independent accelerations. Finding the value of $\ddot{\mathbf{q}}$ in Eqs. (2.2)-(2.3):

$$\begin{Bmatrix}\ddot{\mathbf{q}}^d\\ \ddot{\mathbf{q}}^i\end{Bmatrix} = \begin{bmatrix}\mathbf{\Phi}_\mathbf{q}^d & \mathbf{\Phi}_\mathbf{q}^i\\ \mathbf{0}_{f\times m} & \mathbf{I}_f\end{bmatrix}^{-1}\begin{Bmatrix}\mathbf{c}\\ \ddot{\mathbf{z}}\end{Bmatrix} = \begin{bmatrix}\mathbf{S} & \mathbf{R}\end{bmatrix}\begin{Bmatrix}\mathbf{c}\\ \ddot{\mathbf{z}}\end{Bmatrix} = \mathbf{R}\ddot{\mathbf{z}} + \mathbf{S}\mathbf{c} \tag{2.10}$$

and substituting in the first part of Eq. (2.8):

$$\mathbf{R}^T\mathbf{M}\mathbf{R}\ddot{\mathbf{z}} = \mathbf{R}^T\mathbf{F} - \mathbf{R}^T\mathbf{M}\mathbf{S}\mathbf{c} \tag{2.11}$$

These dynamic equations are completed with the relationship between dependent and independent velocities (similar to Eq. (2.10)), which can be obtained from Eq. (1.3):

$$\dot{\mathbf{q}} = \mathbf{R}\dot{\mathbf{z}} + \mathbf{S}\mathbf{b} \tag{2.12}$$

Products $\mathbf{Sb}$ and $\mathbf{Sc}$ in Eqs. (2.12) and (2.10) can be obtained without calculating matrix $\mathbf{S}$. For instance, $\mathbf{Sc}$ in (2.10) is $\ddot{\mathbf{q}}$ when $\ddot{\mathbf{z}} = \mathbf{0}$. This can be easily calculated from Eq. (2.2). Equation (2.11) can also be considered as a step in the solution of Eq. (1.5). The accelerations $\ddot{\mathbf{q}}$ can be computed from Eq. (2.10) and the Lagrange multipliers from Eq. (2.9).

By defining a state vector $\mathbf{y}^T = \{\dot{\mathbf{z}}^T, \mathbf{q}^T\}^T$, Eqs. (2.11)-(2.12) allow $\dot{\mathbf{y}}$ to be calculated, which is given to the numerical integrator of the motion differential equations. Maggi's method formulated this way is numerically more stable than the methods based on Eqs. (1.5) or (2.8), and is smaller in size. After mentioning these advantages, Laulusa and Bauchau, 2008, also examined its possible drawbacks:

- The choice of independent coordinates is not unique, as a particular set of independent coordinates which is appropriate for one position can become inappropriate for a different one. In this case, it is necessary to stop the integration, choose a new set of independent coordinates, and resume it.
- In every step of the numerical integration, a basis $\mathbf{R}$ for the null space of $\mathbf{\Phi_q}$ has to be evaluated, which can be very expensive for large problems.

– The computation of matrix $\mathbf{R}$ must be carried out in a robust and efficient way. It can be computed by using methods based on the Gauss elimination (or the LU factorization) or with more stable but more expensive methods such as the QR factorization or the singular value decomposition. According to experience, the Gauss elimination and related methods are stable and precise enough for most practical applications. In the modular implementation of algorithms described in this Thesis, it is a relatively simple task to replace the LU factorization, for instance, with the QR one.

These drawbacks, as explained next, can be overcome.

2.2 Double-step Maggi's formulation

The computation of matrix $\mathbf{R}$ in Eq. (2.5) can be very expensive for medium- to large-size multibody systems, particularly if the dynamic equations in Eq. (1.1) are first formulated in Cartesian coordinates (natural or reference point). The necessary computations to determine matrix $\mathbf{R}$ and to evaluate the product of matrices $\mathbf{R}^T\mathbf{M}\mathbf{R}$ in Eq. (2.11) are cumbersome, even with the most careful and efficient numerical implementations.

Some authors such as Negrut, Serban and Potra, 1997, and Rodríguez et al., 2004, have emphasized the advantages of using joint formulations (or relative coordinates) in order to reduce the size of the numerical problem. However, they point out that dynamic formulations with joint coordinates are complex. Nearly all dynamic formulations based on joint coordinates start by opening the closed loops and first considering an open-chain system. The relative coordinates in an open-chain system are independent, so the kinematic and dynamic formulations are simpler at this stage. In a second phase, the closure-of-the-loop constraints is enforced.

The key point of this formulation is to apply Maggi's formulation (2.11) to the Cartesian dynamic equation (1.1) in a simple and efficient way. The matrix transformation of Eq. (2.11) is applied in two steps. First, a transformation from Cartesian to relative open-chain coordinates is applied. This transformation considerably reduces the size of the problem, and it can be carried out analytically in a simple recursive way. Next, a second transformation leads from the open-chain relative coordinates to closed-chain independent relative coordinates. This transformation is carried out numerically, but with a far smaller system than the initial one. This second transformation benefits greatly from the use of linear algebra subroutines.

The first semi-recursive formulation based on a velocity transformation between Cartesian and relative velocities was due to Jerkovsky, 1978. These ideas were subsequently extended by authors such as Kim and Vanderploeg, 1986. More

recently, semi-recursive formulations have been developed by authors like Negrut, Serban and Potra, 1997, and Kim, 2002. In the next section, a new simple and general variant will be described. This formulation is simpler than others found in the bibliography, in certain aspects.

2.2.1 Recursive open-loop equations

The present method starts with the dynamic equations set in Cartesian coordinates, and then applies two velocity transformations that lead to the differential equations of motion using a set of independent relative coordinates.

It is possible to consider only revolute and prismatic joints, because other joints with more degrees of freedom can be decomposed into a combination of revolute and prismatic joints with massless intermediate bodies. See Figure 2.2 for a schematic depiction of a revolute joint and a prismatic joint.

If the system has closed loops, it is first transformed into an open-chain system through the cut-joint method or, in some cases, by removing some bodies with a particular geometry and mass distribution (*rods*). Initially, the motion equations are formulated in Cartesian coordinates. Then, a first velocity transformation switches from the Cartesian velocities to the relative velocities corresponding to the open-chain system.

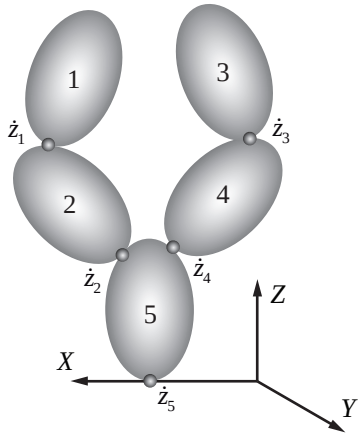


Figure 2.1: Multibody system with tree structure.

In this formulation, the geometry of each moving body is defined in a local reference frame attached to it by using natural coordinates (see García de Jalón et al., 1987), i.e., by defining a set of points and unit vectors that describe the geometry of the body and its joints. In this way, the geometry becomes simpler and cleaner than using multiple *markers* or additional reference frames attached to the moving bodies. When needed, this geometric information is easily transformed to the global reference frame using the body position variables, which are the position vector of the origin of the moving reference frame $\mathbf{r}_i$ and the transformation matrix $\mathbf{A}_i$.

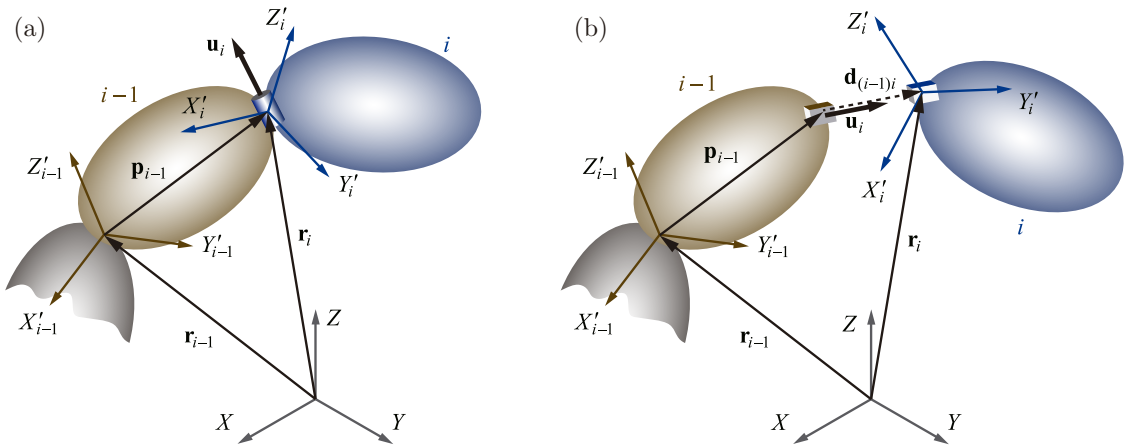


Figure 2.2: (a) Revolute joint. (b) Prismatic joint.

For reasons that will be discussed later, the body's Cartesian velocities $\mathbf{Z}_i$ include the velocity $\dot{\mathbf{s}}_i$ of the point attached to body i that instantaneously coincides with the origin of the inertial reference frame. These Cartesian velocities and accelerations are also used by Negrut, Serban and Potra, 1997, and Kim, 2002. Cartesian velocities and accelerations are then defined, respectively, by

$$\mathbf{Z}_i \equiv \begin{Bmatrix} \dot{\mathbf{s}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix} \tag{2.13}$$

$$\dot{\mathbf{Z}}_i \equiv \begin{Bmatrix} \ddot{\mathbf{s}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} \tag{2.14}$$

Vectors $\mathbf{Z}$ and $\dot{\mathbf{Z}}$ are, respectively, the vectors that contain the Cartesian velocities and accelerations of all bodies:

$$\mathbf{Z}^T = \begin{Bmatrix} \mathbf{Z}_1^T & \mathbf{Z}_2^T & \cdots & \mathbf{Z}_n^T \end{Bmatrix} \tag{2.15}$$

$$\dot{\mathbf{Z}}^T = \begin{Bmatrix} \dot{\mathbf{Z}}_1^T & \dot{\mathbf{Z}}_2^T & \cdots & \dot{\mathbf{Z}}_n^T \end{Bmatrix} \tag{2.16}$$

Using points and unit vectors, joints between contiguous bodies are very easily modeled. For instance, in a revolute joint between bodies i–1 and i (see Figure 2.2(a)), an output point and a unit vector of element i–1 coincide with the input point and unit vector of element i, respectively. For a prismatic joint, both elements share a unit vector, and the input point of element i is located on the line defined by the output point and unit vector of element i–1 (see Figure 2.2(b)); in this case, both elements share the same transformation matrix.

For the dynamics, it is simpler to use expressions for the Cartesian velocities $\mathbf{Y}$ and accelerations $\dot{\mathbf{Y}}$ based on the center of gravity $\mathbf{g}_i$, which are defined as:

$$\mathbf{Y}_i = \begin{Bmatrix} \dot{\mathbf{g}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix} = \begin{bmatrix} \mathbf{I}_3 & -\tilde{\mathbf{g}}_i \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{s}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix} = \mathbf{D}_i \mathbf{Z}_i \tag{2.17}$$

$$\dot{\mathbf{Y}}_i = \begin{Bmatrix} \ddot{\mathbf{g}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} = \begin{bmatrix} \mathbf{I}_3 & -\tilde{\mathbf{g}}_i \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{s}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} + \begin{Bmatrix} \tilde{\boldsymbol{\omega}}_i \tilde{\boldsymbol{\omega}}_i \mathbf{g}_i \\ \mathbf{0} \end{Bmatrix} = \mathbf{D}_i \dot{\mathbf{Z}}_i + \mathbf{e}_i \tag{2.18}$$

Equations (2.17) and (2.18) constitute the definition of matrix $\mathbf{D}_i$ and vector $\mathbf{e}_i$. In these expressions, $\tilde{\mathbf{g}}_i$ and $\tilde{\boldsymbol{\omega}}_i$ are the skew-symmetric matrices associated with vectors $\mathbf{g}_i$ and $\boldsymbol{\omega}_i$, so that for a generic vector $\mathbf{x}$, $\tilde{\mathbf{g}}_i \mathbf{x} = \mathbf{g}_i \times \mathbf{x}$ and $\tilde{\boldsymbol{\omega}}_i \mathbf{x} = \boldsymbol{\omega}_i \times \mathbf{x}$.

For open-chain systems, the Cartesian positions, velocities, and accelerations can be recursively computed upwards from the relative coordinates, velocities, and accelerations. The recursive calculations for positions are straightforward and have been omitted here because they are not important for the first velocity transformation. For velocities and accelerations:

$$\mathbf{Z}_i = \mathbf{Z}_{i-1} + \mathbf{b}_i \dot{z}_i \tag{2.19}$$

$$\dot{\mathbf{Z}}_i = \dot{\mathbf{Z}}_{i-1} + \mathbf{b}_i \ddot{z}_i + \mathbf{d}_i \tag{2.20}$$

where z_i are the relative coordinates, and vectors $\mathbf{b}_i$ and $\mathbf{d}_i$ have simple expressions that depend on the kind of joint i. Note that, if different reference points are used for bodies i and i–1, Eqs. (2.19) and (2.20) should include a transformation matrix $\mathbf{B}_i$. The simplicity of Eqs. (2.19) and (2.20) has important advantages in some subsequent *accumulated expressions*.

For open-chain systems, a velocity transformation defined by the following equation can be set directly:

$$\dot{\mathbf{q}} = \mathbf{R}_1 \dot{z}_1 + \mathbf{R}_2 \dot{z}_2 + \ldots + \mathbf{R}_n \dot{z}_n = \mathbf{R} \dot{\mathbf{z}} \tag{2.21}$$

In this case, column j of matrix $\mathbf{R}$ can be computed directly because its elements are the Cartesian velocities of the bodies that are upwards in the tree, caused by a unit relative velocity in the joint j and with null relative velocities in the remaining joints. Since all the bodies share the same reference point, all have the same velocity $\mathbf{b}_i$, according to Eq. (2.19).

This velocity transformation and the way the system topology is taken into account is better explained with an example. Figure 2.1 shows an open-chain, tree-configured multibody system. As suggested by Negrut, Serban and Potra, 1997, the bodies have been numbered from the leaves to the root, in such a way that each body's number is lower than its parent's. This numbering avoids the later fill-in in the Gauss elimination process. Each body has the same number as its input joint. In this example, the velocity transformation matrix corresponding to Eq. (2.21) has the following form:

$$\mathbf{R}=\begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2 & 0 & 0 & \mathbf{b}_5\\ 0 & \mathbf{b}_2 & 0 & 0 & \mathbf{b}_5\\ 0 & 0 & \mathbf{b}_3 & \mathbf{b}_4 & \mathbf{b}_5\\ 0 & 0 & 0 & \mathbf{b}_4 & \mathbf{b}_5\\ 0 & 0 & 0 & 0 & \mathbf{b}_5\end{bmatrix}=\begin{bmatrix}\mathbf{I} & \mathbf{I} & 0 & 0 & \mathbf{I}\\ 0 & \mathbf{I} & 0 & 0 & \mathbf{I}\\ 0 & 0 & \mathbf{I} & \mathbf{I} & \mathbf{I}\\ 0 & 0 & 0 & \mathbf{I} & \mathbf{I}\\ 0 & 0 & 0 & 0 & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{b}_1 & 0 & 0 & 0 & 0\\ 0 & \mathbf{b}_2 & 0 & 0 & 0\\ 0 & 0 & \mathbf{b}_3 & 0 & 0\\ 0 & 0 & 0 & \mathbf{b}_4 & 0\\ 0 & 0 & 0 & 0 & \mathbf{b}_5\end{bmatrix}\equiv\mathbf{T}\mathbf{R}_d \quad (2.22)$$

where $\mathbf{I}$ is the identity matrix of size 6×6, $\mathbf{T}$ is the *path matrix* that defines the connectivity of the multibody system, and $\mathbf{R}_d$ is a diagonal matrix whose elements are the vectors $\mathbf{b}_i$ defined in Eq. (2.19). Remember that vector $\mathbf{b}_i$ represents the velocity of the point that coincides with the inertial frame origin, induced by a unit relative velocity in joint i.

The introduction of the path matrix $\mathbf{T}$ is a key point of this formulation. This allows the topology to be considered in a straightforward way. Other authors need to introduce complicated expressions to explain the recursive processes on different branches that start from a common junction body. Observe that the k row of the path matrix $\mathbf{T}$ defines the joints or relative coordinates that are below body k, while column k contains the bodies that are upwards of joint k.

Taking into account Eqs. (2.17) and (2.18), the virtual power of the inertia and external forces acting on the whole system is:

$$\begin{aligned}\sum_{i=1}^{n}\mathbf{Y}_i^{*T}\left(\mathbf{M}_i\dot{\mathbf{Y}}_i-\mathbf{Q}_i\right)=\sum_{i=1}^{n}\mathbf{Z}_i^{*T}\mathbf{D}_i^T\left(\mathbf{M}_i\mathbf{D}_i\dot{\mathbf{Z}}_i+\mathbf{M}_i\mathbf{e}_i-\mathbf{Q}_i\right)=\\ =\sum_{i=1}^{n}\mathbf{Z}_i^{*T}\left(\bar{\mathbf{M}}_i\dot{\mathbf{Z}}_i-\bar{\mathbf{Q}}_i\right)=0\end{aligned} \quad (2.23)$$

where the virtual velocities have been denoted with an asterisk (*). The matrices in Eq. (2.23) are:

$$\mathbf{M}_i=\begin{bmatrix}m_i\mathbf{I_3} & \mathbf{0}\\ \mathbf{0} & \mathbf{J}_i\end{bmatrix} \quad (2.24)$$

$$\bar{\mathbf{M}}_i=\mathbf{D}_i^T\mathbf{M}_i\mathbf{D}_i=\begin{bmatrix}m_i\mathbf{I_3} & -m_i\tilde{\mathbf{g}}_i\\ m_i\tilde{\mathbf{g}}_i & \mathbf{J}_i-m_i\tilde{\mathbf{g}}_i\tilde{\mathbf{g}}_i\end{bmatrix} \quad (2.25)$$

$$\bar{\mathbf{Q}}_i=\mathbf{D}_i^T\left(\mathbf{M}_i\mathbf{e}_i-\mathbf{Q}_i\right) \quad (2.26)$$

where m_i is the mass of body i and $\mathbf{J}_i$ is its inertia tensor. Matrix $\mathbf{D}_i$ and vector $\mathbf{e}_i$ are defined in Eq. (2.18). In Eqs. (2.23) and (2.24), the inertia matrices and vector forces denoted with an upper bar refer to the origin of the global reference frame.

By defining the global system inertia matrix, $\bar{\mathbf{M}}$, the force vector, $\bar{\mathbf{Q}}$ and the acceleration vector, $\dot{\mathbf{Z}}$, in the form:

$$\bar{\mathbf{M}} \equiv \mathrm{diag}\left(\bar{\mathbf{M}}_1, \bar{\mathbf{M}}_2, ..., \bar{\mathbf{M}}_n\right) \tag{2.27}$$

$$\bar{\mathbf{Q}}^T = \left[\bar{\mathbf{Q}}_1^T, \bar{\mathbf{Q}}_2^T, ..., \bar{\mathbf{Q}}_n^T\right] \tag{2.28}$$

$$\dot{\mathbf{Z}}^T = \left[\dot{\mathbf{Z}}_1^T, \dot{\mathbf{Z}}_2^T, .., \dot{\mathbf{Z}}_n^T\right] \tag{2.29}$$

the dynamic Eqs. (2.23) can be written as:

$$\mathbf{Z}^{*T}\left(\bar{\mathbf{M}}\dot{\mathbf{Z}} - \bar{\mathbf{Q}}\right) = 0 \tag{2.30}$$

Using Eq. (2.22) for matrix $\mathbf{R}$, the velocity transformation and its time derivative can be written for the whole system as:

$$\mathbf{Z} = \mathbf{R}\dot{\mathbf{z}} = \mathbf{T}\mathbf{R}_d\dot{\mathbf{z}} \tag{2.31}$$

$$\dot{\mathbf{Z}} = \mathbf{T}\mathbf{R}_d\ddot{\mathbf{z}} + \mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}} \tag{2.32}$$

Substituting Eqs. (2.31) and (2.32) in Eq. (2.30) and taking into account that the relative virtual velocities are independent, a set of equations analogous to Eq. (2.11) is obtained:

$$\mathbf{R}_d^T\left(\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\right)\mathbf{R}_d\ddot{\mathbf{z}} = \mathbf{R}_d^T\mathbf{T}^T\left(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}}\right) \tag{2.33}$$

It is interesting to visualize the pattern of the inertia matrix in Eq. (2.33), for the open-chain example in Figure 2.1:

$$\begin{aligned}\mathbf{R}_d^T\mathbf{M}^\Sigma\mathbf{R}_d \equiv \mathbf{R}_d^T\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\mathbf{R}_d = \\ = \begin{bmatrix} \mathbf{b}_1^T\mathbf{M}_1^\Sigma\mathbf{b}_1 & \mathbf{b}_1^T\mathbf{M}_1^\Sigma\mathbf{b}_2 & \mathbf{0} & \mathbf{0} & \mathbf{b}_1^T\mathbf{M}_1^\Sigma\mathbf{b}_5 \\ \mathbf{b}_2^T\mathbf{M}_1^\Sigma\mathbf{b}_1 & \mathbf{b}_2^T\mathbf{M}_2^\Sigma\mathbf{b}_2 & \mathbf{0} & \mathbf{0} & \mathbf{b}_2^T\mathbf{M}_2^\Sigma\mathbf{b}_5 \\ \mathbf{0} & \mathbf{0} & \mathbf{b}_3^T\mathbf{M}_3^\Sigma\mathbf{b}_3 & \mathbf{b}_3^T\mathbf{M}_3^\Sigma\mathbf{b}_4 & \mathbf{b}_3^T\mathbf{M}_3^\Sigma\mathbf{b}_5 \\ \mathbf{0} & \mathbf{0} & \mathbf{b}_4^T\mathbf{M}_3^\Sigma\mathbf{b}_3 & \mathbf{b}_4^T\mathbf{M}_4^\Sigma\mathbf{b}_4 & \mathbf{b}_4^T\mathbf{M}_4^\Sigma\mathbf{b}_5 \\ \mathbf{b}_5^T\mathbf{M}_1^\Sigma\mathbf{b}_1 & \mathbf{b}_5^T\mathbf{M}_2^\Sigma\mathbf{b}_2 & \mathbf{b}_5^T\mathbf{M}_3^\Sigma\mathbf{b}_3 & \mathbf{b}_5^T\mathbf{M}_4^\Sigma\mathbf{b}_4 & \mathbf{b}_5^T\mathbf{M}_5^\Sigma\mathbf{b}_5 \end{bmatrix}\end{aligned} \tag{2.34}$$

where

$$\begin{aligned}\mathbf{M}_1^\Sigma &= \bar{\mathbf{M}}_1 \\ \mathbf{M}_2^\Sigma &= \bar{\mathbf{M}}_2 + \mathbf{M}_1^\Sigma \\ \mathbf{M}_3^\Sigma &= \bar{\mathbf{M}}_3 \\ \mathbf{M}_4^\Sigma &= \bar{\mathbf{M}}_4 + \mathbf{M}_3^\Sigma \\ \mathbf{M}_5^\Sigma &= \bar{\mathbf{M}}_5 + \mathbf{M}_2^\Sigma + \mathbf{M}_4^\Sigma\end{aligned} \tag{2.35}$$

Matrices $\mathbf{M}_i^\Sigma$ are *accumulated inertia matrices*, as described by many authors. They represent the accumulation of the inertia matrices of all the elements that are upwards of joint *i*. Observe that the pattern of the inertia matrix in Eq. (2.34) symmetrically reproduces the pattern of path matrix $\mathbf{T}$ in Eq. (2.22). In an analogous way, the accumulated external forces $\mathbf{Q}^\Sigma$ and accumulated velocity-dependent inertia forces $\mathbf{P}^\Sigma$ can be computed from the r.h.s. of Eq. (2.33):

$$\mathbf{Q}^{\Sigma} \equiv \mathbf{T}^T \bar{\mathbf{Q}} = \begin{Bmatrix} \mathbf{Q}_1^{\Sigma} \\ \mathbf{Q}_2^{\Sigma} \\ \mathbf{Q}_3^{\Sigma} \\ \mathbf{Q}_4^{\Sigma} \\ \mathbf{Q}_5^{\Sigma} \end{Bmatrix} \tag{2.36}$$

$$\mathbf{P}^{\Sigma} \equiv -\mathbf{T}^T \bar{\mathbf{M}} \mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} = \begin{Bmatrix} \mathbf{P}_1^{\Sigma} \\ \mathbf{P}_2^{\Sigma} \\ \mathbf{P}_3^{\Sigma} \\ \mathbf{P}_4^{\Sigma} \\ \mathbf{P}_5^{\Sigma} \end{Bmatrix} \tag{2.37}$$

where

$$\begin{aligned}
\mathbf{Q}_1^{\Sigma} &= \bar{\mathbf{Q}}_1 & \mathbf{P}_1^{\Sigma} &= -\bar{\mathbf{M}}_1 \left(\mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} \right)_1 \\
\mathbf{Q}_2^{\Sigma} &= \bar{\mathbf{Q}}_2 + \mathbf{Q}_1^{\Sigma} & \mathbf{P}_2^{\Sigma} &= -\bar{\mathbf{M}}_2 \left(\mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} \right)_2 + \mathbf{P}_1^{\Sigma} \\
\mathbf{Q}_3^{\Sigma} &= \bar{\mathbf{Q}}_3 & \mathbf{P}_3^{\Sigma} &= -\bar{\mathbf{M}}_3 \left(\mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} \right)_3 \\
\mathbf{Q}_4^{\Sigma} &= \bar{\mathbf{Q}}_4 + \mathbf{Q}_3^{\Sigma} & \mathbf{P}_4^{\Sigma} &= -\bar{\mathbf{M}}_4 \left(\mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} \right)_4 + \mathbf{P}_3^{\Sigma} \\
\mathbf{Q}_5^{\Sigma} &= \bar{\mathbf{Q}}_5 + \mathbf{Q}_2^{\Sigma} + \mathbf{Q}_4^{\Sigma} & \mathbf{P}_5^{\Sigma} &= -\bar{\mathbf{M}}_5 \left(\mathbf{T} \dot{\mathbf{R}}_d \dot{\mathbf{z}} \right)_5 + \mathbf{P}_2^{\Sigma} + \mathbf{P}_4^{\Sigma}
\end{aligned} \tag{2.38}$$

The meaning of the accumulation of external forces is clear. With respect to the velocity-dependent inertia forces, Eq. (2.38) shall be related to Eq. (2.32).

The matrix in Eq. (2.34) shows the advantages of numbering the bodies and joints from the leaves to the root: the Gaussian elimination (or the LU factorization) keeps the pattern of zeroes in the matrix, i.e., it maintains the skyline or sparsity of this matrix, avoiding some arithmetic operations.

Equations (2.33) constitute a system of ODEs whose coefficient matrix and right-hand side vector can be computed recursively in a very efficient way.

2.2.2 Coordinate partitioning and Maggi's approach

The dynamics of closed-chain multibody systems can be formulated by adding the constraint equations to the dynamic equations (2.33) corresponding to the open-chain system. It is then possible to select an independent subset of relative coordinates, in such a way that a set of ODEs will be obtained at the end. This is carried out by a new velocity transformation similar to the one in Eq. (2.11).

In this case, the transformation matrix $\mathbf{R}_z$ will be obtained numerically from the Jacobian matrix of the loop-closing constraint equations. In many applications, it is possible to find a set of independent relative coordinates valid for the whole motion range (Serban and Haug, 2000), as is the case in this Thesis.

Kinematic constraints For the sake of simplicity, the closure-of-the-loop constraint equations are first formulated in Cartesian coordinates and then transformed to relative coordinates. In this work, two ways to set the closed-chain constraint equations are considered. The first one is the cut-joint method, which is very common in the literature. The cutting of a revolute joint (or a spherical one) defined with natural coordinates will be examined in detail. The second method involves the elimination of one or more *rods* (slender bodies with two spherical joints and a negligible moment of inertia around the direction of the axis). This second procedure is rarely found in the bibliography, and it is particularly interesting in applications such as car suspension systems, where rods are very common.

Rods are difficult to analyze: they are sometimes considered as distance constraints, neglecting their inertia forces. Some commercial packages do not even allow the introduction of bodies with two spherical joints, and require the replacement of a spherical joint by a universal joint, so that the free rotation around the rod axis is eliminated.

If the rod is removed and enforced using constraints, only one constant distance constraint is needed. On the other hand, if the rod is kept as a body and the loop is opened by cutting one of the end spherical joints, three constraint equations and two relative coordinates (of the universal joint) are needed. The first approach is therefore more efficient and is the one used hereby, even though the exact inclusion of its inertia properties might be a bit more cumbersome.

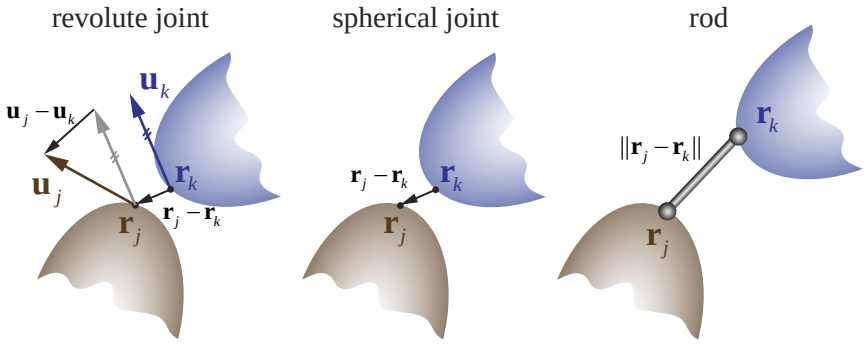


Figure 2.3: Revolute joint, spherical joint and rod constraint equations.

Figure 2.3(a) shows a revolute joint defined in natural coordinates. In order to formulate the constraints of this joint, two points and two unit vectors from different bodies are forced to coincide. Accordingly, the constraint equations are:

$$\mathbf{\Phi}^R \equiv \begin{Bmatrix} \mathbf{r}_j - \mathbf{r}_k \\ \mathbf{u}_j - \mathbf{u}_k \end{Bmatrix} = \mathbf{0} \quad \text{(5 independent equations)} \tag{2.39}$$

In contrast, the spherical joint only requires three constraint equations:

$$\mathbf{\Phi}^S \equiv \mathbf{r}_j - \mathbf{r}_k = \mathbf{0} \qquad \text{(3 independent equations)} \tag{2.40}$$

For the rod element in Figure 2.3(c), only one constant distance condition is necessary, which can be written compactly as:

$$\Phi^{\text{rod}} \equiv (\mathbf{r}_j - \mathbf{r}_k)^T(\mathbf{r}_j - \mathbf{r}_k) - l_{jk}^2 = 0 \tag{2.41}$$

The constraint equations (2.39)-(2.41) can be expressed in terms of the relative coordinates $\mathbf{z}$. This is not difficult, because points $\mathbf{r}_j$ and $\mathbf{r}_k$, and unit vectors $\mathbf{u}_j$ and $\mathbf{u}_k$ can be expressed as functions of the relative coordinates of the joints in their respective branches of the open-chain system.

It is also necessary to compute the Jacobian matrix of constraints (2.39)-(2.41) with respect to relative coordinates $\mathbf{z}$. As the aforesaid constraints are expressed as a function of Cartesian coordinates, the chain derivative rule can be used. For instance, for the constant distance constraint (2.41):

$$\mathbf{\Phi}_{\mathbf{z}} = \mathbf{\Phi}_{\mathbf{r}_j}\frac{\partial \mathbf{r}_j}{\partial \mathbf{z}} + \mathbf{\Phi}_{\mathbf{r}_k}\frac{\partial \mathbf{r}_k}{\partial \mathbf{z}} = \mathbf{\Phi}_{\mathbf{r}_j}\frac{\partial \dot{\mathbf{r}}_j}{\partial \dot{\mathbf{z}}} + \mathbf{\Phi}_{\mathbf{r}_k}\frac{\partial \dot{\mathbf{r}}_k}{\partial \dot{\mathbf{z}}} \tag{2.42}$$

The derivatives with respect to the coordinates $\mathbf{r}_j$ and $\mathbf{r}_k$ in Eq. (2.41) are:

$$\mathbf{\Phi}_{\mathbf{r}_j} = 2(\mathbf{r}_j^T - \mathbf{r}_k^T) \tag{2.43}$$

$$\mathbf{\Phi}_{\mathbf{r}_k} = -2(\mathbf{r}_j^T - \mathbf{r}_k^T) \tag{2.44}$$

The derivatives of the position vectors $\mathbf{r}_j$ and $\mathbf{r}_k$ with respect to the relative coordinates $\mathbf{z}$ can be computed from the velocities of these points induced by unit relative velocities in the joints between the fixed body and bodies j and k, respectively. For instance, if the joint i is a revolute joint determined by a point $\mathbf{r}_i$ and a unit vector $\mathbf{u}_i$, located between the junction body and point $\mathbf{r}_j$, the velocity of point j originated by a unit relative velocity in joint i can be expressed as:

$$\frac{\partial \dot{\mathbf{r}}_j}{\partial \dot{z}_i} = \mathbf{u}_i \times (\mathbf{r}_j - \mathbf{r}_i) = \tilde{\mathbf{u}}_i(\mathbf{r}_j - \mathbf{r}_i) \tag{2.45}$$

If joint i were a prismatic joint also determined by $\mathbf{r}_i$ and $\mathbf{u}_i$, the derivative would be (unit translation motion):

$$\frac{\partial \dot{\mathbf{r}}_j}{\partial \dot{z}_i} = \mathbf{u}_i \tag{2.46}$$

In any case, it can be assumed that the closure-of-the-loop constraint equations $\mathbf{\Phi}$ and their Jacobian matrix $\mathbf{\Phi}_{\mathbf{z}}$ are either known or easy to compute. Using the coordinate partitioning method based on Gaussian elimination with full pivoting as in Eq. (2.5), the following partitioned velocity equation follows:

$$\begin{bmatrix}\mathbf{\Phi}_{\mathbf{z}}^d & \mathbf{\Phi}_{\mathbf{z}}^i\end{bmatrix}\begin{Bmatrix}\dot{\mathbf{z}}^d \\ \dot{\mathbf{z}}^i\end{Bmatrix} = \mathbf{0} \quad \rightarrow \quad \dot{\mathbf{z}}^d = -\left(\mathbf{\Phi}_{\mathbf{z}}^d\right)^{-1}\mathbf{\Phi}_{\mathbf{z}}^i\dot{\mathbf{z}}^i \tag{2.47}$$

where it is assumed that matrix $\mathbf{\Phi}_{\mathbf{z}}^{d}$ is invertible and that the constraint equations are holonomic and scleronomic. Eq. (2.47) allows an easy calculation of the transformation matrix $\mathbf{R}_{\mathbf{z}}$ that relates dependent and independent relative velocities, in the form:

$$\dot{\mathbf{z}} = \mathbf{R}_{\mathbf{z}} \dot{\mathbf{z}}^{i} \tag{2.48}$$

$$\begin{Bmatrix} \dot{\mathbf{z}}^{d} \\ \dot{\mathbf{z}}^{i} \end{Bmatrix} = \begin{bmatrix} -\left(\mathbf{\Phi}_{\mathbf{z}}^{d}\right)^{-1} \mathbf{\Phi}_{\mathbf{z}}^{i} \\ \mathbf{I} \end{bmatrix} \dot{\mathbf{z}}^{i} \tag{2.49}$$

$$\mathbf{R}_{\mathbf{z}} = \begin{bmatrix} -\left(\mathbf{\Phi}_{\mathbf{z}}^{d}\right)^{-1} \mathbf{\Phi}_{\mathbf{z}}^{i} \\ \mathbf{I} \end{bmatrix} \tag{2.50}$$

If this equation is differentiated with respect to time, the following holds:

$$\ddot{\mathbf{z}} = \mathbf{R}_{\mathbf{z}} \ddot{\mathbf{z}}^{i} + \dot{\mathbf{R}}_{\mathbf{z}} \dot{\mathbf{z}}^{i} \tag{2.51}$$

The velocity transformation defined by Eqs. (2.48) and (2.51) is introduced in the equations of motion (2.33). The final Maggi's equation is obtained by premultiplication by matrix $\mathbf{R}_{\mathbf{z}}^{T}$:

$$\mathbf{R}_{\mathbf{z}}^{T} \mathbf{R}_{d}^{T} \mathbf{M}^{\Sigma} \mathbf{R}_{d} \mathbf{R}_{\mathbf{z}} \ddot{\mathbf{z}}^{i} = \mathbf{R}_{\mathbf{z}}^{T} \mathbf{R}_{d}^{T} \mathbf{Q}^{\Sigma} - \mathbf{R}_{\mathbf{z}}^{T} \mathbf{R}_{d}^{T} \mathbf{M}^{\Sigma} \left(\dot{\mathbf{R}}_{d} \dot{\mathbf{z}} + \mathbf{R}_{d} \dot{\mathbf{R}}_{\mathbf{z}} \dot{\mathbf{z}}^{i} \right) \tag{2.52}$$

All the terms in this equation are known, except for the parenthesis containing the derivatives of the transformation matrices. To obtain them, it is simpler to compute the two terms together. Considering Eq. (2.48), the parenthesis in Eq. (2.52) can be written as:

$$\begin{aligned} \dot{\mathbf{R}}_{d} \dot{\mathbf{z}} + \mathbf{R}_{d} \dot{\mathbf{R}}_{\mathbf{z}} \dot{\mathbf{z}}^{i} = & \\ &= \dot{\mathbf{R}}_{d} \mathbf{R}_{\mathbf{z}} \dot{\mathbf{z}}^{i} + \mathbf{R}_{d} \dot{\mathbf{R}}_{\mathbf{z}} \dot{\mathbf{z}}^{i} = \\ &= \left(\dot{\mathbf{R}}_{d} \mathbf{R}_{\mathbf{z}} + \mathbf{R}_{d} \dot{\mathbf{R}}_{\mathbf{z}} \right) \dot{\mathbf{z}}^{i} = \\ &= \frac{d\left(\mathbf{R}_{d} \mathbf{R}_{\mathbf{z}}\right)}{dt} \dot{\mathbf{z}}^{i} \end{aligned} \tag{2.53}$$

This derivative can be computed from the product of velocity transformations that relates Cartesian and independent relative velocities:

$$\mathbf{Z} = \mathbf{R} \dot{\mathbf{z}} = \mathbf{T} \mathbf{R}_{d} \dot{\mathbf{z}} = \mathbf{T} \mathbf{R}_{d} \mathbf{R}_{\mathbf{z}} \dot{\mathbf{z}}^{i} \tag{2.54}$$

Taking the time derivative of this equation:

$$\dot{\mathbf{Z}} = \mathbf{T} \mathbf{R}_{d} \mathbf{R}_{\mathbf{z}} \ddot{\mathbf{z}}^{i} + \mathbf{T} \frac{d\left(\mathbf{R}_{d} \mathbf{R}_{\mathbf{z}}\right)}{dt} \dot{\mathbf{z}}^{i} \tag{2.55}$$

In this equation, the product of the path matrix $\mathbf{T}$ times the sought derivative, can be computed numerically as the Cartesian accelerations $\dot{\mathbf{Z}}$ that would be

produced by the true velocities $\dot{\mathbf{z}}$ and null relative independent accelerations ($\ddot{\mathbf{z}}^i = \mathbf{0}$):

$$\mathbf{R}_\mathbf{z}^T \mathbf{R}_d^T \mathbf{M}^\Sigma \mathbf{R}_d \mathbf{R}_\mathbf{z} \ddot{\mathbf{z}}^i = \mathbf{R}_\mathbf{z}^T \mathbf{R}_d^T \mathbf{Q}^\Sigma - \mathbf{R}_\mathbf{z}^T \mathbf{R}_d^T \mathbf{T}^T \bar{\mathbf{M}} \underbrace{\mathbf{T} \frac{d\left(\mathbf{R}_d \mathbf{R}_\mathbf{z}\right)}{dt} \dot{\mathbf{z}}^i}_{\text{numerically computed}} \tag{2.56}$$

Thus, a way to compute the terms in the Maggi's set of ODEs (2.52) has been completed. Grouping some of the terms, the equation can be rewritten as:

$$\hat{\mathbf{M}}(\mathbf{z}) \ddot{\mathbf{z}}^i(t) = \hat{\mathbf{Q}}(t, \mathbf{z}, \dot{\mathbf{z}}) - \hat{\mathbf{P}}(t, \mathbf{z}, \dot{\mathbf{z}}) \tag{2.57}$$

Two velocity transformations have been introduced. The first one, from Cartesian to open-chain relative velocities, is applied directly and leads to an accumulation of forces and inertias. The second one is applied to a (usually) smaller system in a fully numerical way.

Equation (2.57) can be integrated over time using an appropriate time-integration scheme. Maggi's formulation implies the definition of the state vector $\mathbf{y}^T = \{\mathbf{z}^T, \dot{\mathbf{z}}^{iT}\}$ instead of the somewhat more typical $\mathbf{y}^T = \{\mathbf{z}^{iT}, \dot{\mathbf{z}}^{iT}\}$. The latter requires, in each time-step, the solution of a nonlinear system of equations (the finite displacement problem) in order to compute the dependent positions ($\mathbf{z}$) from the independent ones ($\mathbf{z}^i$), whereas that process is avoided by the former, resulting in a higher efficiency. On the other hand, Maggi's approach suffers from a numerical drift coming from the fact that the position-level constraints are not enforced. This drawback can be overcome by controlling the error in the constraints and re-computing the dependent positions from the independent ones when the error exceeds a tolerance. In the models analyzed in this Thesis, the variation of the total energy of the system, even in long (30+ s) simulations, has found to be minimal.

Rod's inertia For the sake of brevity, the details about the inertia forces of the rods that have been removed to open the closed loops have been omitted. This information is detailed in other sources (Rodríguez et al., 2004). It is enough to point out that a rod introduces coupling terms between the inertia forces of the two branches connected by it. The topological information needed to compute these coupling terms is also contained in the path matrix.

2.3 Implementation

The efficient solution of the motion differential equations (2.57) requires the use of a robust time integration algorithm and a careful implementation of linear algebra subroutines. Some details on how this is achieved are provided next.

2.3.1 Time integration

Let $\mathbf{y}$ be a vector containing the current system states, and $\dot{\mathbf{y}}$ its time derivative. The general form of vector ODEs is:

$$\begin{cases} \dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \\ \mathbf{y}(t = 0) = \mathbf{y}_0 \end{cases} \tag{2.58}$$

which, in the present case, takes the form

$$\begin{cases} \dot{\mathbf{y}} \equiv \begin{Bmatrix} \dot{\mathbf{z}} \\ \ddot{\mathbf{z}}^i \end{Bmatrix} = \begin{Bmatrix} \dot{\mathbf{z}} \\ \hat{\mathbf{M}}(\mathbf{z})^{-1} \left[\hat{\mathbf{Q}}(t, \mathbf{z}, \dot{\mathbf{z}}) - \hat{\mathbf{P}}(t, \mathbf{z}, \dot{\mathbf{z}}) \right] \end{Bmatrix} \\ \mathbf{y}(t = 0) = \mathbf{y}_0 \end{cases} \tag{2.59}$$

There is a great variety of time integration algorithms for the solution of DAEs and ODEs. Depending on the type of system under study, the form of the equations, and the numerical requirements, different integrators can be used.

The two basic families of integrators are explicit and implicit methods. When the system is stiff (as it is the case with vehicles), forces in very different scales occur. In these cases, explicit schemes require short time-steps to achieve stability, and thus result in an accurate but slow integration. The new state of the system only depends on past states, and therefore can be calculated directly:

$$\mathbf{y}_{i+1} = \mathbf{f}(t_i, \mathbf{y}_i) \tag{2.60}$$

On the other hand, implicit integrators are less accurate but faster, as they can use longer time-steps while keeping the integration stable. Implicit schemes always require some sort of iteration, because the new system state depends both on new and past states:

$$\mathbf{y}_{i+1} = \mathbf{f}(t_i, \mathbf{y}_i, t_{i+1}, \mathbf{y}_{i+1}) \tag{2.61}$$

Also, integrators can use a uniform time grid (constant step size integrators) or a variable time grid (variable step size integrators). When the simulation is numerically demanding at certain moments and steady the rest of the time, variable step size integrators can make the most of those situations. In turn, constant step size integrators have to use the most demanding time-step of the simulation. In this Thesis, only constant time-step integrators have been used.

In the context of vehicle dynamics, some examples of state-of-the-art integrators are HHT, Newmark, Adams-Bashforth-Moulton and Runge-Kutta schemes. In this Thesis, the implementation of sensitivity analysis and optimization techniques was more relevant than the computational efficiency of forward dynamics. As it will be presented later, AD tools and multi-objective optimization algorithms make the integration process more complicated. For this reasons, it was decided to use a sufficiently efficient and stable constant time-step integra-

tor: 4th order Runge-Kutta. Experience has shown that this integrator provides a good trade-off between efficiency and ease of implementation in the particular field of vehicle dynamics.

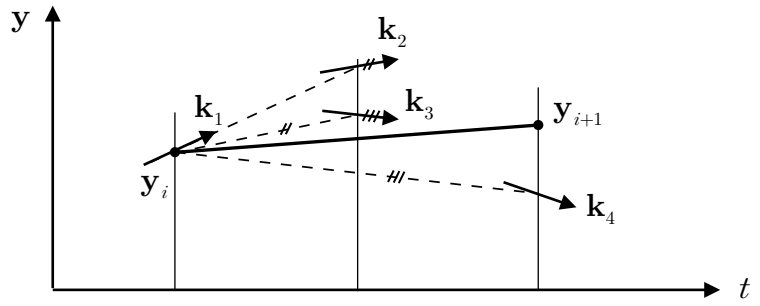


Figure 2.4: Representation of Runge-Kutta function evaluations.

Following the 4th order Runge-Kutta integrator, the expression for the new state depends on four evaluations of the state vector derivative as:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{2.62}$$

$$\mathbf{k}_1 = \mathbf{f}(t_i, \mathbf{y}_i) \tag{2.63}$$

$$\mathbf{k}_2 = \mathbf{f}(t_i + \frac{h}{2}, \mathbf{y}_i + \frac{h}{2}\mathbf{k}_1) \tag{2.64}$$

$$\mathbf{k}_3 = \mathbf{f}(t_i + \frac{h}{2}, \mathbf{y}_i + \frac{h}{2}\mathbf{k}_2) \tag{2.65}$$

$$\mathbf{k}_4 = \mathbf{f}(t_i + h, \mathbf{y}_i + h\mathbf{k}_3) \tag{2.66}$$

Figure 2.4 shows a 2D representation of the four evaluations of function $\mathbf{f}$, whose weighted addition provides the value of the new state vector. In order to get the sense of how the integrator works, the mathematical flow of the integrator is plotted on the left part of Figure 2.5, and the state vector derivative flow on the right part. Vector $\mathbf{y}^T = \{\mathbf{z}^T, \dot{\mathbf{z}}^{iT}\}$ is the state vector; $\mathbf{y}_i$ is the state vector at time t_i; $\mathbf{Y}$ is the matrix of state vectors and h is the time-step. The rest of the variables have already been defined.

In spite of the high number of function evaluations per numerical integration and the short time-steps required, the 4th order Runge-Kutta integrator and the double-step Maggi's formulation constitute a robust and efficient formulation. More information on efficient implementations of this formulation and others can be found at Hidalgo, 2013.

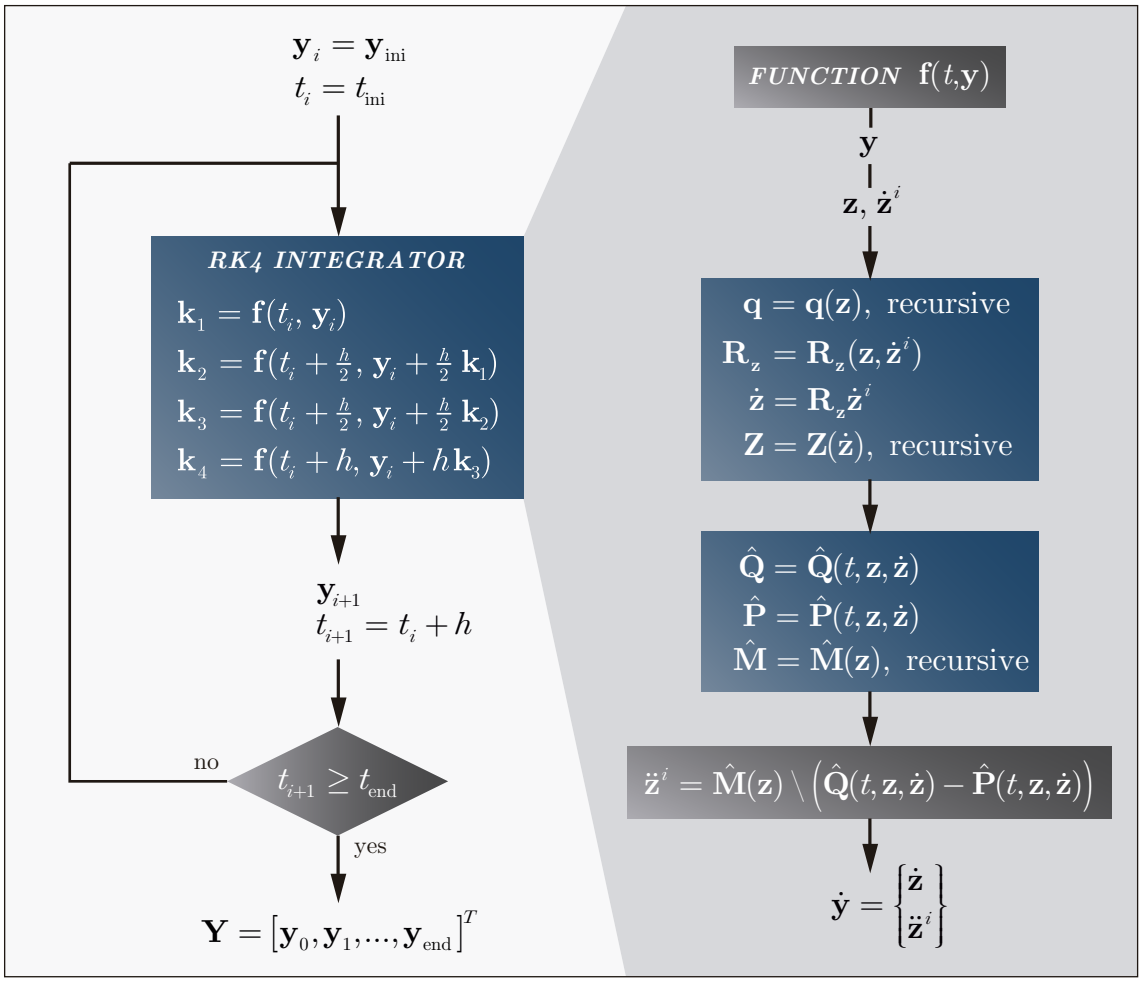


Figure 2.5: Integrator and state vector derivative flows.

2.3.2 Linear algebra subroutines

The simulation software has been implemented both in MATLAB and C/C++. This strategy has interesting advantages because it makes the most of both programming languages. The MATLAB module is used for four main tasks:

- Definition and debugging of new models
- Prototyping of new algorithms
- Optimization loop (Chapter 6)
- Postprocessing

On the other hand, C/C++ language is used for two key tasks:

- Vehicle forward dynamics (Chapter 3)
- 3D graphics viewer (Section 2.5)
- Sensitivity analysis (Chapter 5)

The modularity of the code is essential for a good maintenance and for a fast development of models and algorithms. The code is general-purpose, meaning

that very different kinds of multibody systems can be simulated with it. The input of the simulation is a datafile with structures defining the topology, the simulation characteristics, the specific functions defining the kinematic guidance of coordinates, and the user-defined external forces.

Simulations can be run from MATLAB (in which case the C/C++ computation of forward dynamics is called through a MEX-function) or as a standalone C/C++ simulation. The advantage of calling the program from MATLAB is that the simulation results are returned to MATLAB and can be used in an outer loop (for instance an optimization loop) or post-processed. In these cases, the overhead caused by calling the C/C++ code through a MEX-function has found to be small, especially compared to the time taken by the forward dynamics or the sensitivity analysis themselves. Nevertheless, standalone simulations are also useful for demonstration purposes and for human-in-the-loop (HITL) simulations. As far as C/C++ compilers are concerned, Microsoft Visual C++ 9.0 and Intel Compiler have been tested, but no significant differences in the computation times have been found. Therefore the former is used throughout (see Appendix A).

The double-step Maggi's formulation is a very efficient way of computing the null-space basis of $\mathbf{\Phi_q}$. However, it requires a careful implementation of linear algebra solvers to achieve real-time efficiency. Basic linear algebra subprograms have been implemented to improve the efficiency of linear algebra calculations. The most expensive steps in the numerical algorithm are the computation of matrix $\mathbf{R_z}$ from Eq. (2.48) and the product of matrices to arrive at Eq. (2.52) from Eq. (2.33). These operations can benefit from the use of specialized routines such as dense BLAS or sparse matrix functions.

The product of matrices is simple and does not deserve further attention. More important is the LU factorization of the Jacobian matrix $\mathbf{\Phi_z}$ and the subsequent solution steps to get matrix $\mathbf{R_z}$ according to Eq. (2.48). The presence of redundant constraints, which leads to systems of redundant but compatible equations, is a source of difficulties. The sparse function MA48 from Harwell is able to deal with this, but there is not an equivalent function for dense matrices in LAPACK. Thus, two C/C++ functions have been built. They directly call the BLAS functions `cblas_idamax`, `cblas_dswap` and `cblas_dger` to perform the LU factorization with column pivoting and row swapping, and then solve two triangular systems of linear equations. These operations have been carried out through rank-1 matrix updating (function `cblas_dger`), which is crucial to gain efficiency.

Integrator	Evaluations	Elapsed time (s)	Time per evaluation (s)
Adams/Newmark	2530	2.11	8.34×10^{-4}
Adams/HHT	2455	2.22	9.04×10^{-4}
MBS3D/RK4	28000	2.40	8.56×10^{-5}
Adams/GSTIFF	2100	2.95	1.40×10^{-3}
Adams/WSTIFF	2162	5.96	2.76×10^{-3}

Table 2.1: Adams times (10-ms time-step).

Integrator	Evaluations	Elapsed time (s)	Time per evaluation (s)
MBS3D/RK4	28000	2.40	8.56×10^{-5}
Adams/Newmark	3994	3.87	9.69×10^{-4}
Adams/HHT	4305	4.02	9.34×10^{-4}
Adams/GSTIFF	3416	6.06	1.77×10^{-3}
Adams/WSTIFF	3108	10.10	3.25×10^{-3}

Table 2.2: Adams times (5-ms time-step).

Integrator	Evaluations	Elapsed time (s)	Time per evaluation (s)
MBS3D/RK4	28000	2.40	8.56×10^{-5}
Adams/GSTIFF	3416	3.82	1.12×10^{-3}
Adams/Newmark	3994	3.99	9.99×10^{-4}
Adams/HHT	4305	4.02	9.34×10^{-4}
Adams/WSTIFF	3108	7.34	2.36×10^{-3}

Table 2.3: Adams times (5-ms time-step and 4-thread parallelization).

2.3.3 Benchmark

The proposed solver scheme (double-step Maggi's formulation with 4^{th} order Runge-Kutta integrator) has been developed and enhanced over the past years at INSIA's Computational Mechanics Group. In order to illustrate its performance, a small benchmark has been carried out. The model used as a benchmark is a coach model fully detailed in the following chapters. Without anticipating further details, here the coach is simulated while undergoing a double lane-change maneuver (see Chapter 3). The exact same model under the exact same external forces is run both in Adams 2012 and MBS3D (see Appendix A for software details), and the dynamic simulation is timed. In both cases, only the forward dynamics are computed, and data output and graphical display are disabled. According to Adams documentation, C++ executables are more efficient than their Fortran counterparts. Thus, four C++ integrators (namely

Newmark, HHT, GSTIFF and WSTIFF) are run in Adams, using, in each case, the most efficient Adams/Solver formulation.

MBS3D and the selected Adams integrators are quite different from one another. While the former is an explicit constant step size method, the latter are implicit variable step size algorithms. Thus, the way accuracy is enforced differs. A 1-ms time-step is considered in MBS3D. Obviously, for a fair comparison, a longer time-step must be set in Adams. In past investigations with an in-house implicit integrator (see Chapter 4), it was found that the equivalent implicit time-step was around 5 ms. Therefore, Adams simulations are configured with two different (variable) time-steps: 5 ms and 10 ms. The error is set to 10^{-4}. Table 2.1 through Table 2.3 show the elapsed times of the simulations

The results show that MBS3D times are very competitive when compared to those of state-of-the-art commercial algorithms. MBS3D is outperformed by Adams only when the time-step is 10 ms. In conclusion, even though developing efficient methods is not the purpose of this Thesis, it has been demonstrated that MBS3D in its current form can be used to simulate multibody systems efficiently. Due to the fact that optimization procedures require numerous computations of the forward dynamics, MBS3D is going to be used as the starting point for efficient optimization of multibody systems.

2.4 Recursive computation of joint reactions

The forward dynamics problem has been successfully solved in the previous section. The results of the forward analysis are the generalized positions, velocities and accelerations of the system and the value of the applied forces over time. In some cases, this analysis is enough to account for the fundamental behavior of the multibody system and no postprocess computations are required. However, dynamic analyses of real-life mechanical systems often require additional calculations like the solution of the inverse dynamics problem or the computation of joint reactions. The latter is discussed in this section.

2.4.1 Constraint forces

Joint reactions are crucial in the dynamic simulation of mechanical systems in general and vehicle dynamics in particular, since joint forces play a very important role in design optimization. Lagrange multipliers are closely related to joint reactions. As it has concisely been explained in Chapter 1 (see Eq. (1.1)), they can be used to formulate index-3 motion differential equations:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi}_{\mathbf{q}}^{T}\boldsymbol{\lambda} = \mathbf{F} \tag{2.67}$$

where all terms have already been defined. If no redundant constraints are present, the Lagrange multiplier vector can be obtained as:

$$\boldsymbol{\lambda} = (\boldsymbol{\Phi}_{\mathbf{q}}^T)^{-1}(\mathbf{F} - \mathbf{M}\ddot{\mathbf{q}}) \tag{2.68}$$

The physical meaning of Lagrange multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ is linked to the magnitude of the constraint forces acting along the constraint directions. In other words, the product $\boldsymbol{\Phi}_{\mathbf{q}}^T\boldsymbol{\lambda} \in \mathbb{R}^n$ is the vector of constraint forces needed to enforce the constraints over the dynamic simulation. If the rows of the Jacobian matrix $\boldsymbol{\Phi}_{\mathbf{q}} \in \mathbb{R}^{m\times n}$ are of unit Euclidean norm, they can be regarded as the m directions of the constraint forces, and $\boldsymbol{\lambda}$ as a vector containing the m lengths of those vector forces.

As already explained, the presented double-step Maggi's formulation does not follow this constraint enforcement strategy. Instead, a set of ODEs without Lagrange multipliers is integrated. However, once the forward dynamics have been solved, index-1 equations can be written to obtain the values of the Lagrange multipliers, which are necessary for the computation of constraint forces and, eventually, joint reactions. Let us include the Lagrange multipliers corresponding to the closure-of-the-loop constraint equations in the recursive open-loop equations of motion (2.33):

$$\mathbf{R}_d^T\left(\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\right)\mathbf{R}_d\ddot{\mathbf{z}} + \boldsymbol{\Phi}_{\mathbf{z}}^T\boldsymbol{\lambda} = \mathbf{R}_d^T\mathbf{T}^T\left(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}}\right) \tag{2.69}$$

Recalling the coordinate partitioning method already used in Eqs. (2.1) and (2.47), and using relative coordinates:

$$\mathbf{S}_{\mathbf{z}} \equiv \begin{bmatrix} \mathbf{S}_{\mathbf{z}}^d \\ \mathbf{0}_{f\times m} \end{bmatrix} = \begin{bmatrix} \left(\boldsymbol{\Phi}_{\mathbf{z}}^d\right)^{-1} \\ \mathbf{0}_{f\times m} \end{bmatrix} \tag{2.70}$$

$$\mathbf{R}_{\mathbf{z}} \equiv \begin{bmatrix} \mathbf{R}_{\mathbf{z}}^d \\ \mathbf{I}_f \end{bmatrix} = \begin{bmatrix} -\left(\boldsymbol{\Phi}_{\mathbf{z}}^d\right)^{-1}\boldsymbol{\Phi}_{\mathbf{z}}^i \\ \mathbf{I}_f \end{bmatrix} \tag{2.71}$$

where matrix $\boldsymbol{\Phi}_{\mathbf{z}}^d$ is invertible or, at least, of full column rank (in order to have left inverse). Recall that $f = n - r$ is the number of degrees of freedom. Note that the columns of matrix $\mathbf{R}_{\mathbf{z}}$ are a basis for $\ker(\boldsymbol{\Phi}_{\mathbf{z}})$, whereas matrix $\mathbf{S}_{\mathbf{z}}$ is a right inverse of $\boldsymbol{\Phi}_{\mathbf{z}}$. Lagrange multipliers can be found by pre-multiplying Eq. (2.69) by $\mathbf{S}_{\mathbf{z}}^T$ and considering Eq. (2.7):

$$\mathbf{S}_{\mathbf{z}}^T\mathbf{R}_d^T\left(\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\right)\mathbf{R}_d\ddot{\mathbf{z}} + \underbrace{\mathbf{S}_{\mathbf{z}}^T\boldsymbol{\Phi}_{\mathbf{z}}^T}_{\mathbf{I}}\boldsymbol{\lambda} = \mathbf{S}_{\mathbf{z}}^T\mathbf{R}_d^T\mathbf{T}^T\left(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}}\right) \tag{2.72}$$

Then, introducing Eq. (2.51) and grouping terms by means of the same accumulated variables used in previous sections, the following equation can be written:

$$\boldsymbol{\lambda} = \mathbf{S}_{\mathbf{z}}^T\mathbf{R}_d^T\left[\mathbf{Q}^\Sigma - \mathbf{M}^\Sigma\mathbf{R}_d\mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^i - \mathbf{M}^\Sigma\left(\dot{\mathbf{R}}_d\dot{\mathbf{z}} - \mathbf{R}_d\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^i\right)\right] \tag{2.73}$$

Finally, applying the technique explained in Eqs. (2.53)–(2.55) for the computation of the parenthesis term, the final expression is obtained:

$$\boldsymbol{\lambda} = \mathbf{S}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\left[\mathbf{Q}^{\Sigma} - \mathbf{M}^{\Sigma}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^{i} - \mathbf{T}^{T}\bar{\mathbf{M}}\mathbf{T}\frac{d\left(\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\right)}{dt}\dot{\mathbf{z}}^{i}\right] \tag{2.74}$$

Note that this expression has many similarities with Eq. (2.56), and therefore some of the terms for the computation of independent accelerations can be reused for the computation of Lagrange multipliers.

The presence of redundant constraints needs special consideration in the computation of Lagrange multipliers. Contrary to the computation of $\mathbf{R}_{\mathbf{z}}$ in Eqs. (2.48) or (2.71), where redundant constraints are just discarded by the numerical factorization, the computation of $\mathbf{S}_{\mathbf{z}}$ in Eq. (2.70) requires additional care. If the latter is solved through numerical factorization, a set of (redundant) constraints will be numerically discarded in each time-step, and this set will be different each time, depending on the specific values of the Jacobian matrix. Since the factorization process is carried out by an external computer algebra subroutine, it is not easy to know what rows (and therefore, what lambdas) are being discarded.

Instead, only the minimum number of (mathematically independent) constraints are added to the Jacobian matrix, which leads to a computation of non-redundant Lagrange multipliers. However, other redundant constraints coming from the physical nature of the system may be present. An example of an intrinsically overconstrained system is the 3D four-bar mechanism. This system is redundantly constrained even though the user may only introduce the minimum number of constraint equations. In these cases, the solution of Eqs. (2.70) and (2.71) through LU factorization is equivalent to computing the minimum norm solution of the redundant system of equations. This has proven to be a meaningful solution in most situations (García de Jalón and Gutiérrez-López, 2013).

2.4.2 Joint forces

In the presented formulation, one of the key stages is the use of the principle of virtual power to formulate the equations of motion (see Eq. (2.23)). One of the consequences of this step is that the joint forces vanish from the equations, as they do not generate virtual power. The reason for this is that the two forces acting on the two bodies that make up the joint cancel each other out as a result of Newton's third law. For this reason, joint forces are not present in the motion differential equations.

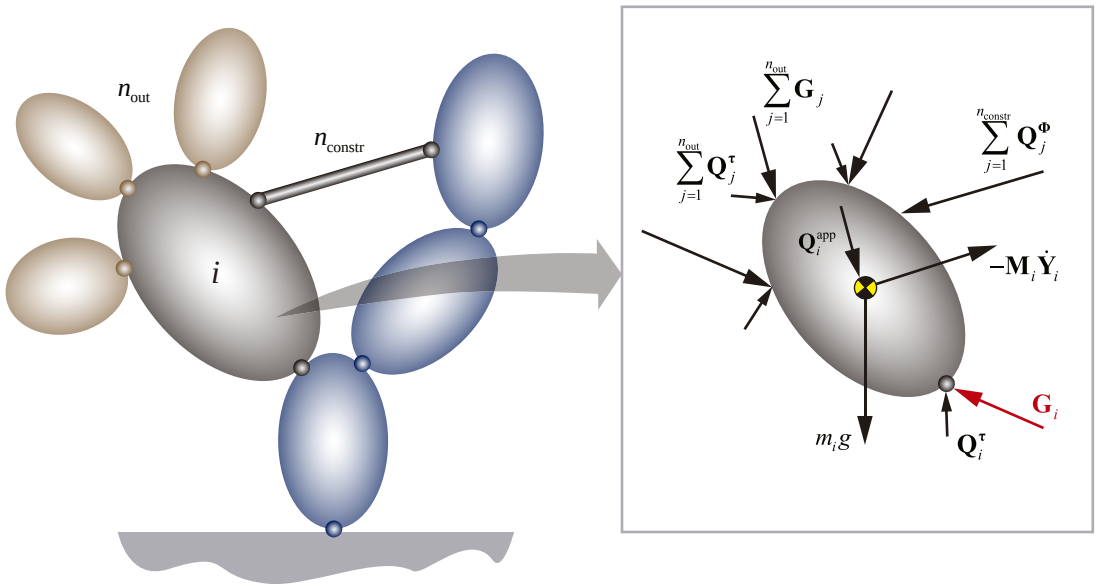


Figure 2.6: Generic free-body diagram for the computation of reactions.

The values of the joint forces must be computed as a postprocess of the numerical integration. Once the constraint forces have been computed, a recursive traversal from the leaves to the roots can be performed so as to compute joint reactions. If the equilibrium of forces is considered on one of the leaf bodies, the only unknown is the reaction with the parent body, since all other forces (constraint, external and inertia forces) are known. The free body diagram of body i is depicted in Figure 2.6. The analytical expression of the equilibrium of forces would be:

$$-\sum_{j=1}^{n_{\text{out}}} \mathbf{Q}_j^{\tau} - \sum_{j=1}^{n_{\text{out}}} \mathbf{G}_j + \sum_{j=1}^{n_{\text{constr}}} \mathbf{Q}_j^{\Phi} + \sum_{j=1}^{n_{\text{rods}}} \mathbf{Q}_j^{\text{rod}} - \mathbf{M}_i\dot{\mathbf{Y}}_i + \mathbf{Q}_i^{\text{app}} + m_i\mathbf{g} + \mathbf{Q}_i^{\tau} + \mathbf{G}_i = \mathbf{0} \quad (2.75)$$

where $\mathbf{Q}_j^{\tau}$ are the n_{out} joint forces applied on the relative coordinates of the output bodies, $\mathbf{G}_j$ are the n_{out} reactions of the output bodies, $\mathbf{Q}_j^{\Phi}$ are the constraint forces, $\mathbf{Q}_j^{\text{rod}}$ are the inertia forces of the n_{rods} rods linked to body i, $\mathbf{M}_i\dot{\mathbf{Y}}_i$ is the inertia force of body i, $\mathbf{Q}_i^{\text{app}}$ are the applied forces, $m_i\mathbf{g}$ is the weight of the body, $\mathbf{Q}_i^{\tau}$ is the joint force applied on the parent joint and $\mathbf{G}_i$ is the reaction on the parent joint. If the system is traversed from the leaves to the root, $\mathbf{G}_i$ is the only unknown of the free body diagram. Note that the terms *force* and *reaction* in this context are 6×1 vectors, each containing a 3×1 linear force and a 3×1 torque vector.

Applied forces can be divided into relative applied forces and Cartesian applied forces, as previously explained. Among the relative applied forces, some originate from the output bodies and some from the input body. If the node has no children, then $n_{\text{out}} = 0$, and the applied forces in the output joints $\mathbf{Q}_j^{\tau}$ (and the reactions of the output bodies $\mathbf{G}_j$) will be null. Otherwise, the node has child

bodies and these forces can be easily computed. The equations for the computation of joint motor forces, depending on the joint type involved, are:

$$\mathbf{Q}^{\boldsymbol{\tau},P} \equiv \begin{Bmatrix} \tau\mathbf{u} \\ \mathbf{r}_{\text{cog}} \times \tau\mathbf{u} \end{Bmatrix} \tag{2.76}$$

$$\mathbf{Q}^{\boldsymbol{\tau},R} \equiv \begin{Bmatrix} \mathbf{0} \\ \tau\mathbf{u} \end{Bmatrix} \tag{2.77}$$

where τ is the value of the force, $\mathbf{u}$ is the direction upon which the prismatic (P) or revolute (R) is defined, and $\mathbf{r}_{\text{cog}}$ is the position of the output joint relative to the COG of body i. The summation of relative applied forces of output bodies $\mathbf{Q}_j^{\boldsymbol{\tau}}$ is known, as is the relative applied force of the input joint $\mathbf{Q}_i^{\boldsymbol{\tau}}$.

Reaction $\mathbf{G}_i$ is the resulting calculation of each free-body equilibrium. As the tree is traversed recursively, reactions $\mathbf{G}_j$ in the output joints will be known.

Constraint forces, as explained, can be computed from the values of the Lagrange multipliers and the directions of the constraints. Assuming that the constraints are of unit length, the expressions for the reaction forces of the spherical joint (S), the revolute joint (R) and the rod are:

$$\mathbf{Q}^{\boldsymbol{\Phi},S} \equiv \begin{Bmatrix} \boldsymbol{\lambda}^S \\ \mathbf{r}_{\text{cog}} \times \boldsymbol{\lambda}^S \end{Bmatrix} \tag{2.78}$$

$$\mathbf{Q}^{\boldsymbol{\Phi},R} \equiv \begin{Bmatrix} \boldsymbol{\lambda}^R_{1:3} \\ \mathbf{r}_{\text{cog}} \times \boldsymbol{\lambda}^R_{1:3} + \mathbf{u} \times \boldsymbol{\lambda}^R_{4:6} \end{Bmatrix} \tag{2.79}$$

$$\mathbf{Q}^{\boldsymbol{\Phi},\text{rod}} \equiv \begin{Bmatrix} \lambda_{\text{rod}}\mathbf{u}_{\text{rod}} \\ \mathbf{r}_{\text{cog}} \times \lambda_{\text{rod}}\mathbf{u}_{\text{rod}} \end{Bmatrix} \tag{2.80}$$

where $\mathbf{r}_{\text{cog}}$ is the position of the joint relative to the COG of body i, $\mathbf{u}_{\text{rod}}$ is the unit vector in the direction of the rod. The Lagrange multipliers correspond to the following constraint equations:

$$\boldsymbol{\Phi}^R_{6\times1} \equiv \begin{Bmatrix} \mathbf{r}_j - \mathbf{r}_k \\ \mathbf{u}_j - \mathbf{u}_k \end{Bmatrix} \quad \rightarrow \quad \boldsymbol{\lambda}^R_{6\times1} \tag{2.81}$$

$$\boldsymbol{\Phi}^S_{3\times1} \equiv \mathbf{r}_j - \mathbf{r}_k \quad \rightarrow \quad \boldsymbol{\lambda}^S_{3\times1} \tag{2.82}$$

$$\Phi^{\text{rod}} \equiv (\mathbf{r}_j - \mathbf{r}_k)^T(\mathbf{r}_j - \mathbf{r}_k) - l_{jk}^2 \quad \rightarrow \quad \lambda^{\text{rod}} \tag{2.83}$$

Note that one of the elements of $\boldsymbol{\lambda}^R_{4:6}$ will be null, as only five constraints are independent in the revolute joint.

As already explained in previous sections, rod elements not only introduce kinematic constraints, but also inertia and force terms coming from their inertia

and the forces applied on their COG. The analytical expression of those forces (see Vidal, 2006) is:

$$\mathbf{Q}^{\text{rod}} \equiv -\begin{Bmatrix} \frac{m}{6}(2\mathbf{I}_3\ddot{\mathbf{r}}_j + \mathbf{I}_3\ddot{\mathbf{r}}_k) \\ \mathbf{r}_{\text{cog}} \times \frac{m}{6}(2\mathbf{I}_3\ddot{\mathbf{r}}_j + \mathbf{I}_3\ddot{\mathbf{r}}_k) \end{Bmatrix} + \begin{Bmatrix} m\mathbf{g} \\ \mathbf{r}_{\text{cog}} \times m\mathbf{g} \end{Bmatrix} + \begin{Bmatrix} \mathbf{f}^{\text{app}} \\ \mathbf{r}_{\text{cog}} \times \mathbf{f}^{\text{app}} \end{Bmatrix} \tag{2.84}$$

where m is the mass of the rod, $\mathbf{r}_j$ and $\mathbf{r}_k$ are the Cartesian positions of the ends of the rod and $\mathbf{r}_{\text{cog}}$ is the position of the end of the rod that is linked to body i, relative to the COG of body i.

Applied forces hereby group the external forces applied on the COG of body i and the velocity-dependent inertia term in the form:

$$\mathbf{Q}_i^{\text{app}} \equiv \begin{Bmatrix} \mathbf{F}_i \\ \mathbf{n}_i - \tilde{\boldsymbol{\omega}}_i \mathbf{J}_i \boldsymbol{\omega}_i \end{Bmatrix} \tag{2.85}$$

where $\mathbf{F}_i$ is the translational external force applied on the COG, $\mathbf{n}_i$ is the externally applied torque, $\boldsymbol{\omega}_i$ is the angular velocity of the body and $\mathbf{J}_i$ is the inertia tensor.

Once all terms of Eq. (2.75) have been computed and reaction $\mathbf{G}_i$ has been found, the same process can be performed on the parent body, and so on. Eventually, the fixed body is reached and all joint reactions are known. The computation of reactions is performed along with the numerical integration of the equations of motion, but it could also be carried out as a post-process. Therefore, the numerical burden caused by the computation of reactions is not considered relevant.

2.4.3 Validation

Three simple examples have been analyzed as a validation of joint reactions, each of them exploring a particular feature of the presented formulation. The reactions are then compared numerically and graphically with a reference commercial package, namely MSC Adams (see Appendix A).

All three examples consist of different assemblies of ideal bars (i.e., bars with negligible moment of inertia around the direction of the axis) linked by ideal revolute and spherical joints. The length of the bars parallel to one of the global axis is 1 m. Each bar has a mass of 1 kg. Gravity acts in the $-Z$-direction. All systems have null initial velocities. Simulations of 2 seconds have been carried out. In the case of the recursive formulation, a time-step of 1 millisecond has been used. In Adams, a maximum error of 10^{-4} is set. Little differences between both sets of results come from the different time discretizations used.

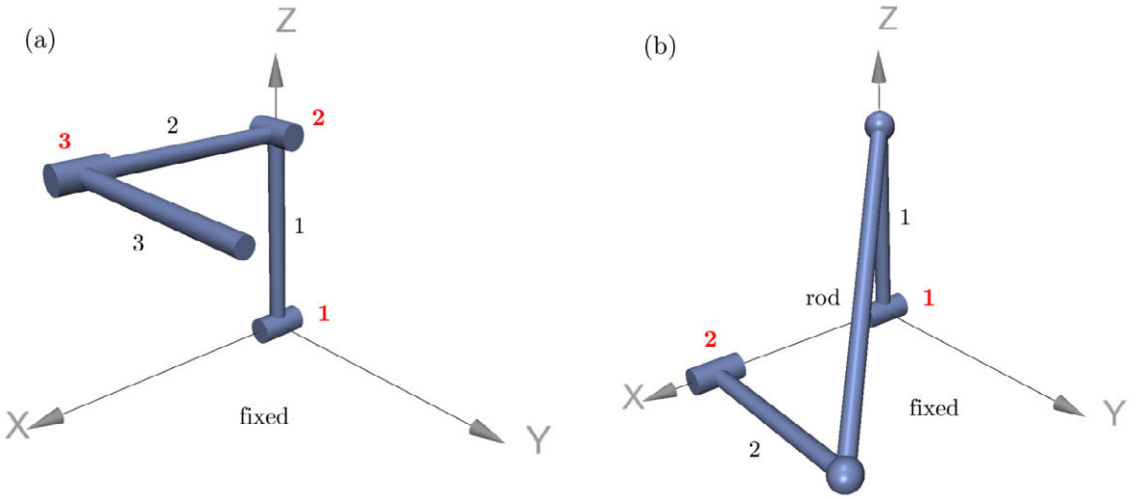


Figure 2.7: Triple-pendulum and four-bar mechanism.

Open-chain triple-pendulum The first example is a 3D triple-pendulum assembled using revolute joints, as Figure 2.7(a) shows. Bodies are numbered with black numbers and joints with red and bold numbers. It is modeled as an open-chain mechanism, and therefore no constraint forces (or Lagrange multipliers) are needed. No external forces other than weight are considered. Reactions in joint 1 are shown in Figure 2.8, both using MBS3D and Adams.

RSSR four-bar mechanism The second example is a 3D four-bar mechanism assembled using revolute and spherical joints. See Figure 2.7(b) for a 3D illustration of the model, where bodies are again numbered in black and joints in red and boldface. The diagonal bar is treated as a rod, therefore a constant distance constraint and the corresponding inertia terms are introduced. This means that there is one constraint force whose value needs to be propagated through the spanning tree. Joint 1 reactions have been plotted in Figure 2.9.

Triple-pendulum modeled as a closed-loop system The basic appearance of this example is exactly the same as the first one. However, the topology is completely different: it is considered as a closed-loop mechanism starting and ending in the fixed element. This requires the introduction of five auxiliary massless elements between the horizontal link and the ground so that six DOFs can be set up between those bodies. As a closed-loop system with no rods, one of the joints needs to be cut and enforced by means of constraint equations; joint 3 is chosen to that end. Additionally, an external force $\mathbf{f} = \{-10, 0, 0\}^T$ [N] acting on the middle point of body 3 is introduced.

The agreement between reactions seems sufficient. In addition to reactions, the optimization of the vehicle dynamic response, like other industrial vehicle dynamics applications, often requires additional simulation capabilities, some of which are summarized in the following sections.

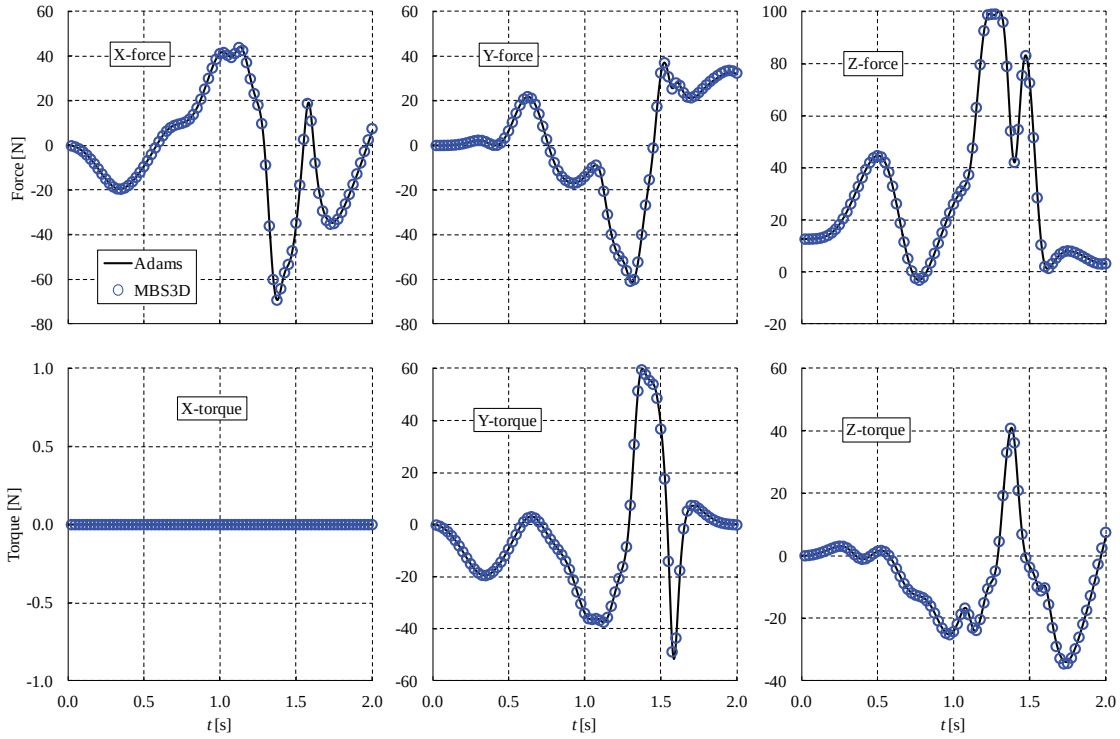


Figure 2.8: Validation of the triple pendulum's reactions on joint 1.

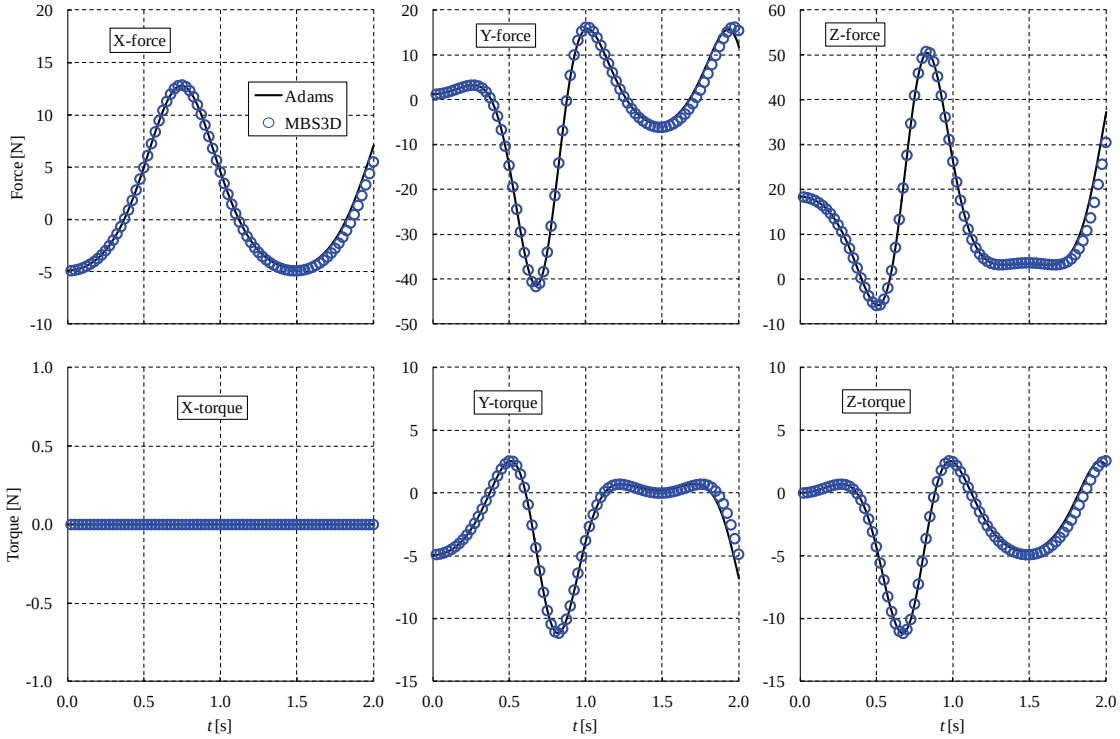


Figure 2.9: Validation of the four-bar mechanism's reactions on joint 1.

2.5 Efficient three-dimensional graphics viewer

Graphical tools for the analysis of the simulation results, according to the experience of other authors (Etman, 1997, Haug, Choi and Komkov, 1986, Erdman, 1995) is essential. Within the simulation of mechanical systems and vehicle dynamics, realistic graphic viewers are very helpful in the design stage and for the evaluation of the system behavior. In short, interactivity between the program and the designer increases the chance of arriving to useful results. As part of the Thesis objectives, a 3D graphics viewer has been developed and plugged into MBS3D, with two key goals: the graphical representation of dynamic maneuvers and the comparison of dynamic responses with different suspension setups. To that end, OpenSceneGraph library (Martz, 2007) was used to develop an in-house graphics viewer.

OpenSceneGraph is an open-source, cross-platform graphics toolkit for the development of high-performance graphics applications such as flight simulators, virtual reality and scientific visualization. It is based on the concept of *scene graphs*, providing an object-oriented framework on top of OpenGL. The key strengths of OpenSceneGraph are its performance, scalability and portability.

2.5.1 Management of models and transformations

The management of 3D models revolves around a configuration file called `Config.txt`. In it, the user specifies the names of the bodies that are going to be displayed in the graphics viewer. If the MBS3D body does not exist, the 3D object will remain static along the numerical integration. This is the case of objects like the scenery.

```
SCENERY Model 3DLANDSCAPE.ive -
ROAD Model 3DROAD.ive -
BODYWORK Model 3DBUSCHASSIS.ive -
R_SUPPORT Generate - Blue
R_L_STAB Generate - Green
R_R_STAB Generate - Green
R_L_WHEEL Model 3D_R_L_WHEEL.ive -
```

Figure 2.10: 3D graphics viewer script.

After the body name, the type of 3D model is specified. The two options are to import a 3D model (e.g. from a CAD environment) or to automatically generate a tubular 3D model based on the geometry of the body. In the first case, the next configuration field should be the keyword `Model` and the name of the file (e.g. `*.ive`, `*.osg` or `*.3ds`). In the second case, the keyword `Generate` is introduced and a color is specified.

Figure 2.10 shows an example of configuration file. The first two objects, `SCENERY` and `ROAD`, will remain static along the integration. The bodies `BODYWORK`,

`R_L_WHEEL`, `R_R_WHEEL`, `F_L_WHEEL` and `F_R_WHEEL` will be imported from the corresponding 3D files, and the rest of the objects will be automatically generated. Figure 2.11 shows the differences between an imported model and a generated one.

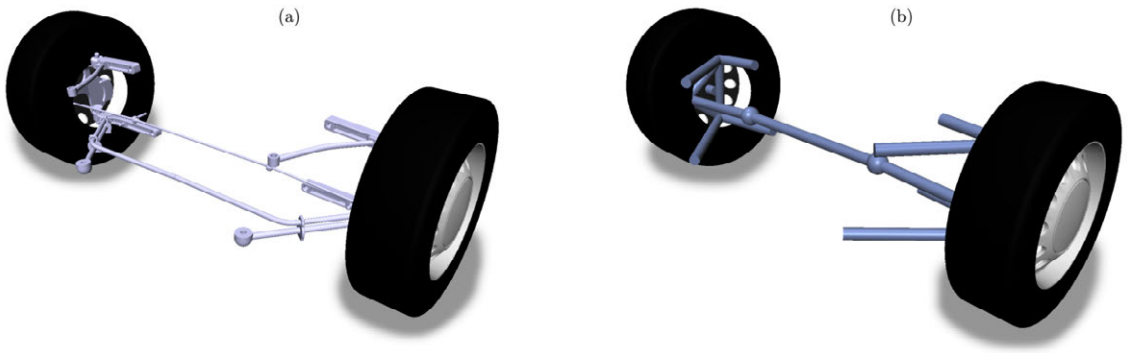


Figure 2.11: (a) Imported and (b) generated front suspension geometry of an IVECO box truck (courtesy of Michele Macchi).

The integrator calls the graphics viewer in every integration step, providing the body positions and rotation matrices. The graphic viewer applies these positions and rotations through function callbacks, which are permanently waiting for new data. As soon as a new set of positions is received, the function callbacks apply them to the graphic objects. Let $\mathbf{r}_b$ be the 3×1 position vector of any reference point of the body, $\mathbf{r}_b^0$ the initial position of the reference point and $\mathbf{A}_b$ the absolute 3×3 body rotation matrix. The position $\mathbf{r}_i$ of a generic point i belonging to the body can be computed as:

$$\mathbf{r}_i = \mathbf{r}_b + \mathbf{A}_b(\mathbf{r}_i^0 - \mathbf{r}_b^0) \tag{2.86}$$

The rotation matrices of the regular bodies are computed by the integrator itself, as part of the recursive computation of positions. However, rods do not belong to the spanning tree and are not treated as regular bodies, thus their rotation matrix has to be computed from the initial and current positions of their ends. This can be achieved by calculating an angle θ and a rotation axis defined by unit vector $\mathbf{u}$, and then applying the Rodrigues' rotation formula:

$$\mathbf{A}_b = \mathbf{I} + \tilde{\mathbf{u}}\tilde{\mathbf{u}}(1-\cos\theta) + \tilde{\mathbf{u}}\sin\theta \tag{2.87}$$

In practice, both the translation and rotation are assembled in a unique 4×4 matrix $\mathbf{M}_b$ that is applied to the entire node geometry. Such matrix can be expressed as the product of three matrices: the first one, containing the initial translation, $\mathbf{r}_b^0$; the second one, containing the current rotation, $\mathbf{A}_b$; and the third one, with the new translation, $\mathbf{r}_b$. These transformations would be equivalent to moving the body to the origin, rotating it and moving it back to the current position, as the following equation shows:

$$\mathbf{M}_b = \mathbf{T}(-\mathbf{r}_b^0)\ \mathbf{R}(\mathbf{A}_b)\ \mathbf{T}(\mathbf{r}_b) = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ -\mathbf{r}_b^{0T} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A}_b & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{r}_b^{T} & 1 \end{bmatrix} \tag{2.88}$$

where $\mathbf{T}$ is the translation transformation and $\mathbf{R}$ the rotation one.

Matrix $\mathbf{M}_b$ is implemented as a `MatrixTransform` object, which is a special node that contains the information of the geometric transformation. This node is part of the scene hierarchy, in a way that all nodes below it are affected by the transformation. Here, positions and rotation matrices are applied to each node independently. This way, the scene tree is simple and each node only depends on one transform node containing the position and rotation data (see Figure 2.12).

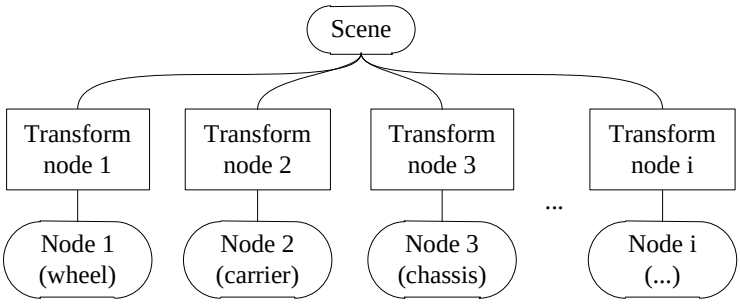


Figure 2.12: Sample scene tree with absolute transformations.

Some of the viewer capabilities are: chassis fill modes (transparent, translucent, solid, etc.), camera modes (top, side, front, travelling, custom, etc.) and motion modes (slow motion, replay, pause). Additionally, more than one vehicle can be displayed in parallel, in order to compare vehicle responses in following chapters, and equally-spaced snapshots of the simulation can be taken in order to present an overview of the system motion in a single picture.

2.5.2 Software architecture and efficiency

There are several ways in which the link between the multibody model and the graphics viewer can be set up. Both in MEX-function and standalone C/C++ implementations, the viewer has been designed to comply with these goals:

- The viewer should not interfere with the numerical integration process, and neither of them should make the other wait unnecessarily. To that end, the viewer is run on a different system thread through the `CreateThread` function of Intel's Threading Building Blocks library (TBB).
- A pipeline based on a *push-pop* structure should be created between both threads, so that data can be exchanged. When a new system position is available, it is pushed into the stack of positions and rotation matrices. In turn, the viewer pulls positions and rotations from the other end.

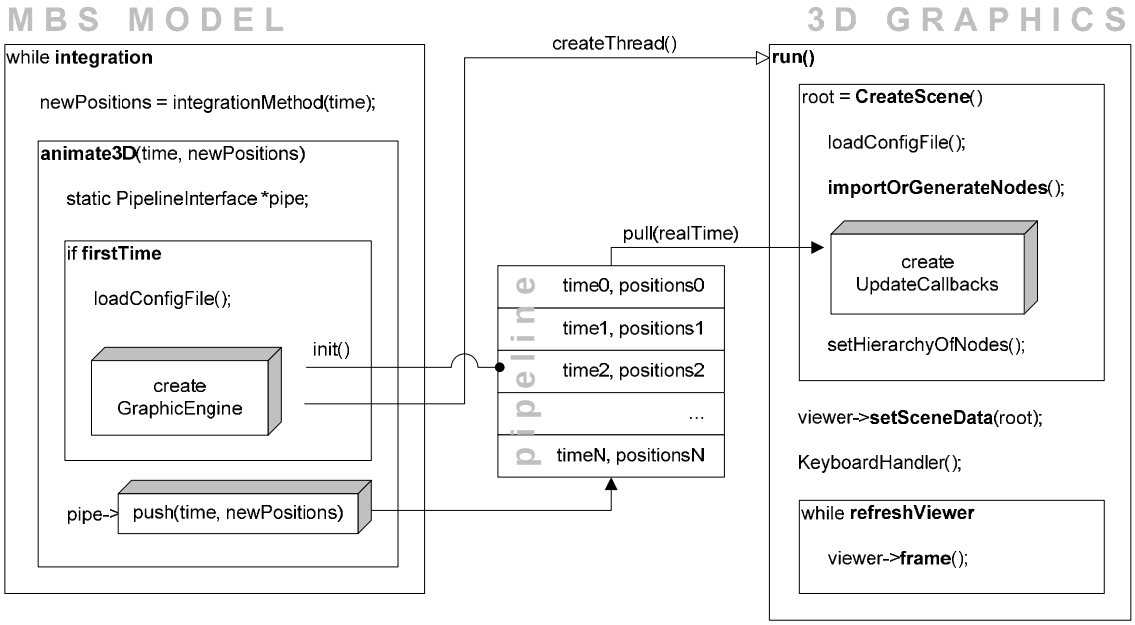


Figure 2.13: Architecture of the graphics viewer.

It is worth stressing the fact that each pair of positions and rotations is always associated with an integrator time, whereas the graphics viewer is based on the processor "real" time. Figure 2.13 shows a diagram of the architecture, where the most important functions and relationships have been highlighted.

The left part, labeled as "MBS model", corresponds to the integration loop of the multibody simulation. Every time a new set of positions `newPositions` is computed, the integrator calls `animate3D`. When first called, it reads the configuration file through the function `loadConfigFile`; then, it creates the pipeline with the `init` function; finally, it launches the graphics viewer by running function `run` in a new thread, which in turn is created via the function `createThread`. The number of accumulated positions in the pipeline will depend on the rate at which positions are pushed and pulled by the integrator and the graphics viewer, respectively.

The right part, labeled as "3D graphics", refers to the graphics viewer, launched by the `run` function. First, the root scene is created in `CreateScene`. Then, the configuration file is read, the nodes are created, and the function callbacks are set up. The callbacks will be in charge of pulling positions from the pipeline through the `pull` function. Certain keyboard events are also defined, in order to allow for camera changes in real time. Finally, the render loop is set up through the `refreshViewer` variable.

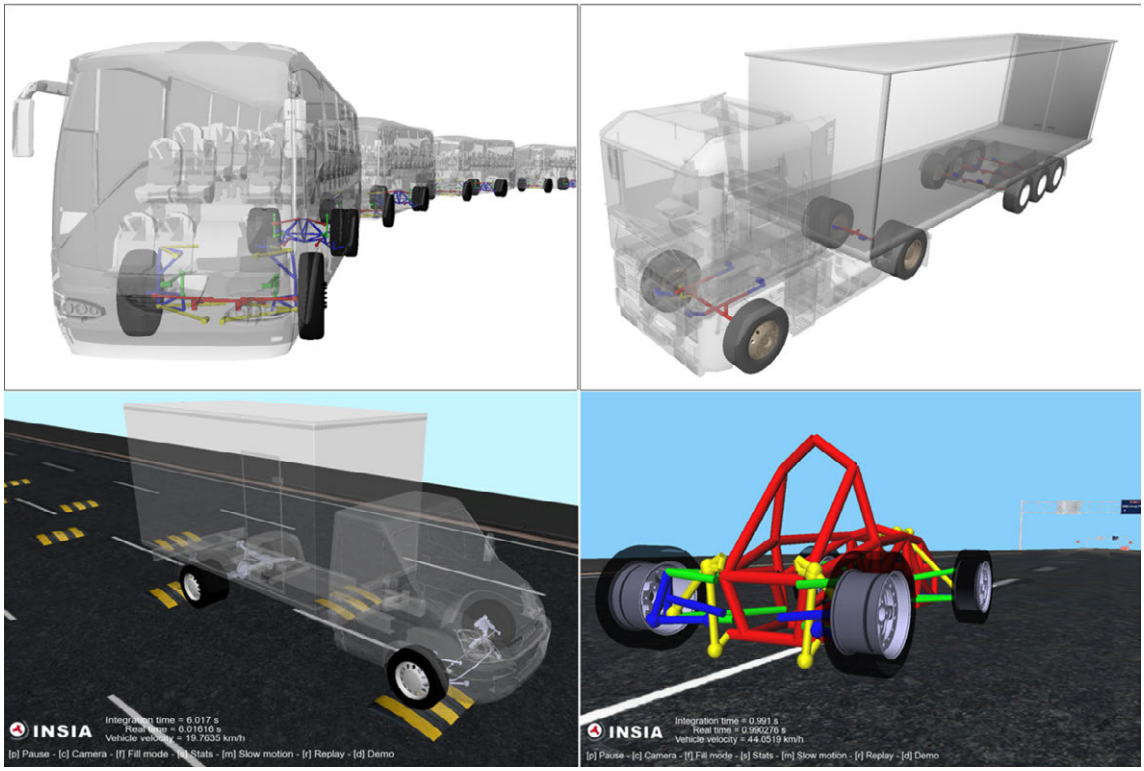


Figure 2.14: Screen captures of the graphic viewer
(coach maneuver, semi-trailer truck, Formula Student racing car and box truck).

Chapter 3

Coach vehicle dynamics

This section presents the coach model whose dynamic response is going to be optimized along the following sections. The model was developed on the basis of a series of tests run with the real coach. Firstly, a detailed explanation of the key elements and modeling assumptions is provided. Then, the dynamic response of the vehicle is analyzed by running virtual tests, both for handling and comfort. It is worth stressing that, since the optimization tool presented in this Thesis deals with the pre-design of vehicle parameters to improve the dynamic behavior, the accuracy of the initial design is not as important as in other applications. The focus will therefore be set in the optimization methodology, rather than in hyper-realistic vehicle modeling.

Figure 3.1: Real coach.

By way of illustration, three recent examples of realistic bus models are cited. Gombor, 2005, presented interesting work where a multibody model of a bus was used to determine the dynamic loads that the suspension exerted on a FEM model of the bus frame, in order to determine the frame stresses. Georgiou, Badarlis and Natsiavas, 2008, developed a high-fidelity flexible multibody model of an urban bus with realistic nonlinear forces, and used it to analyze ride dy-

namics. Sohn et al., 2010, simulated and tested a cruise bus so as to determine the durability of the dampers, achieving a good agreement of results.

3.1 Coach model

The coach under study is a Noge Touring 345 vehicle with frame from Mercedes-Benz. A coordinate-measuring machine has been used on the unloaded coach to obtain global dimensions and the position of key suspension points and joints. A general view of the real coach is shown in Figure 3.1; the general dimensions are displayed in Figure 3.2(a); and an overview of the multibody model is shown in Figure 3.2(b).

The coach has two axles; the front one has two wheels and the rear one has four (assembled as two sets of dual wheels). The total mass of the coach is 13,498 kg. The engine is located longitudinally on the rear end of the coach, behind the rear axle. The main parameters of the vehicle and the multibody model are shown in Table 3.1.

General		**Multibody**	
Parameter	Value	Quantity	Value
Tare weight	13,498 kg	Joints	37
Total length	12 m	Moving bodies	35
Width	2.55 m	Auxiliary bodies	14
Height	3.54 m	Rods	11
Wheelbase	6.25 m	Cartesian coordinates	210
Front axle track	2.05 m	Relative coordinates	35
Rear axle track	1.82 m	Constraint equations	17
Passenger seats	48	Degrees of freedom	18

Table 3.1: Main characteristics of the coach.

Throughout this Thesis, the word *bodywork* is used to name the sprung mass, i.e., the assembly of the superstructure and the frame, including glass windows, seats, etc. Using commercial terminology, the bodywork would be the body, plus the chassis, minus the front and rear axles. A detailed calculation of the bodywork mass is presented later.

Not all vehicle characteristics were available at the design stage. Manufacturers rarely provide component characteristics, and many of them are also impossible to measure. Examples of unknown parameters are the stiffness coefficients of the suspension air springs, the torsion stiffness of the bodywork, the matrix of inertia of the bodywork, the rear support and the uprights, and the damping coefficients of the dampers.

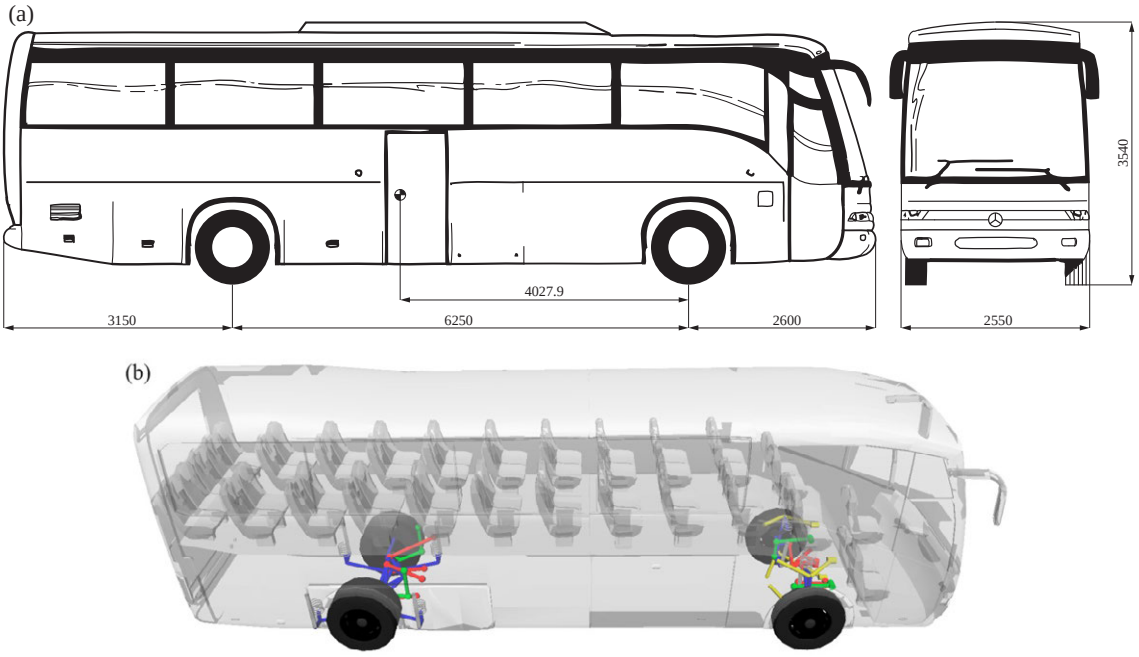


Figure 3.2: (a) Coach dimensions (mm); (b) Coach suspension.

3.1.1 Suspension system

The front suspension system is independent and has two wheels. The system is made up of parallel wishbones (or triangles) between the uprights and the bodywork, an anti-roll (or stabilizer) bar between the uprights, and two dampers along with two air springs between the uprights and the bodywork. The steering system drives each upright by means of a rod. Figure 3.3 shows a view of the front suspension.

The rear suspension system consists of a rigid support containing the solid axle and the differential. The axle has a set of dual wheels on each side. There are four air springs and four dampers between the bodywork and the support, which are also joined to the bodywork through two longitudinal rods and two diagonal rods. An anti-roll bar embedded on the rear support is connected to the right and left sides of the bodywork through a pair of rods. All mass and inertia properties have been measured when possible, and estimated otherwise. Figure 3.4 shows a view of the rear suspension.

A few elements of the suspension such as air springs, dampers and anti-roll bars deserve a further explanation. The mathematical models chosen for these parts are simple enough to not to overload the model with additional nonlinearities and parameter uncertainties, but accurate enough to model the vehicle behavior in standard operating conditions.

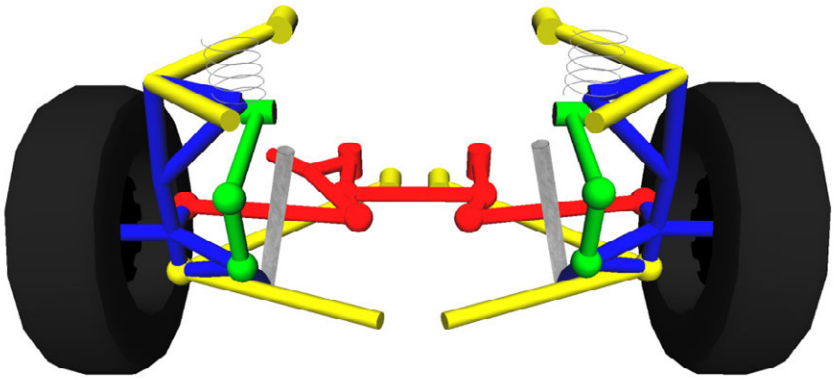


Figure 3.3: Front suspension system: anti-roll bars (green), rods and steering system (red), uprights (blue), wishbones (yellow) and dampers (grey).

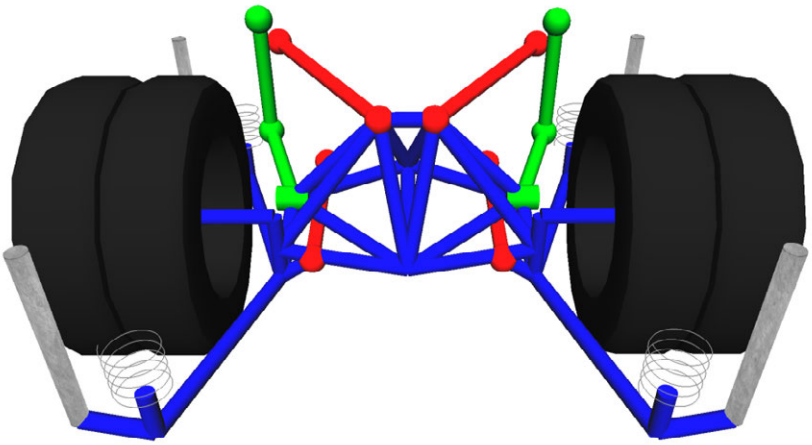
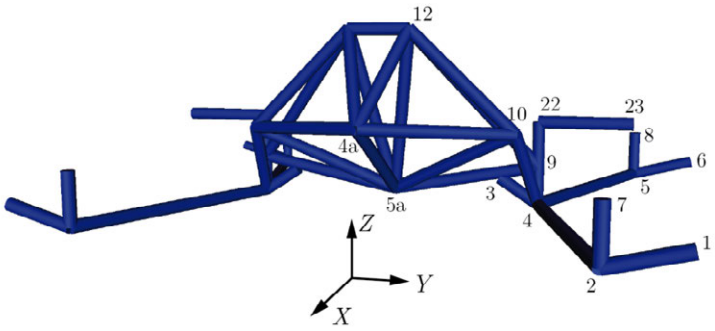
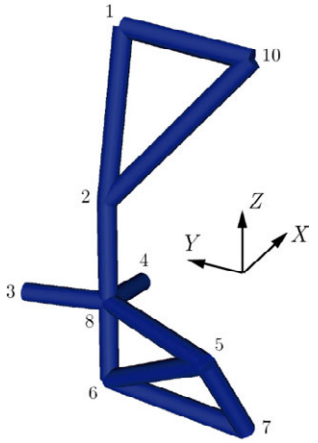


Figure 3.4: Rear suspension system: anti-roll bars (green), rods (red), rear support (blue), and dampers (grey).



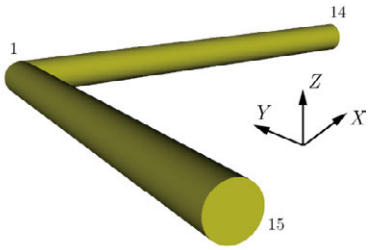
Rear support:
$m = 1{,}062.3$ kg

$$\mathbf{J}^{\text{local}} = 10^{2}\begin{bmatrix}3 & 0 & 0\\ 0 & 3 & 0\\ 0 & 0 & 3\end{bmatrix}\ \text{kg·m}^2$$



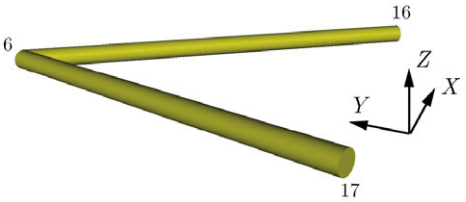
Left upright:
$m = 54.6$ kg

$$\mathbf{J}^{\text{local}} = 10^{-3}\begin{bmatrix}2{,}275 & 0 & 0\\ 0 & 2{,}275 & 0\\ 0 & 0 & 91\end{bmatrix}\ \text{kg·m}^2$$



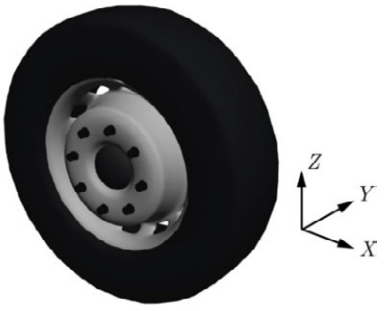
Upper left triangle:
$m = 9.96$ kg

$$\mathbf{J}^{\text{local}} = 10^{-3}\begin{bmatrix}96.5 & 0 & 0\\ 0 & 478.4 & 0\\ 0 & 0 & 572.9\end{bmatrix}\ \text{kg·m}^2$$



Lower left triangle:
$m = 16.73$ kg

$$\mathbf{J}^{\text{local}} = 10^{-3}\begin{bmatrix}705.4 & 0 & 0\\ 0 & 1{,}072 & 0\\ 0 & 0 & 1{,}775\end{bmatrix}\ \text{kg·m}^2$$



Wheel ($\times 6$):
$m = 59.42$ kg

$$\mathbf{J}^{\text{local}} = 10^{2}\begin{bmatrix}3.998 & 0 & 0\\ 0 & 7.133 & 0\\ 0 & 0 & 3.998\end{bmatrix}\ \text{kg·m}^2$$

Figure 3.5: Mass and inertia properties of five key bodies.

Air springs Air (or pneumatic) springs are quite complex to model accurately, as Luque and Mántaras, 2003, explain in detail. The stiffness of the air springs is caused by inside gas pressure, which varies greatly with vehicle load. Moreover, the force characteristic is not linear with respect to the elongation, and has compression and rebound lock-up positions. In what follows, the springs are supposed to be in the loaded coach configuration and within the linear region of the stiffness curve, as Figure 3.6 shows. The X-axis corresponds to the end-to-end elongation of the spring, r_{12}. Note that the force or preload in the initial state, f_0, corresponds to an elongation r_{12}^0.

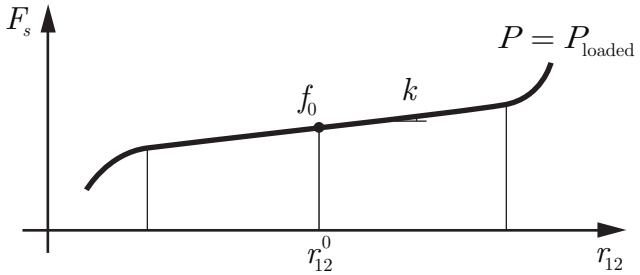


Figure 3.6: Stiffness of the air spring model.

Within the formulation, springs have been modeled as a user force acting on the two end bodies. The magnitude of the force is proportional to the distance between the end points (minus the relaxed length l_0) and the direction is the straight line linking both points. These forces are conveniently translated to the COG of the attached bodies by adding the corresponding torques. The mathematical expression of the spring force is:

$$\mathbf{F}_{s1} = -\mathbf{F}_{s2} = k\big(\|\mathbf{r}_{12}\| - l_0\big)\frac{\mathbf{r}_{12}}{\|\mathbf{r}_{12}\|} = \big(k\|\mathbf{r}_{12}\| + f_0\big)\frac{\mathbf{r}_{12}}{\|\mathbf{r}_{12}\|} \tag{3.1}$$

where $\mathbf{F}_{si}$ is the spring force acting on point i of the body to which the spring is attached; $\mathbf{r}_i$ is the current Cartesian position of point i; l_0 is the relaxed length of the spring; f_0 is the preload force; and k is the stiffness coefficient.

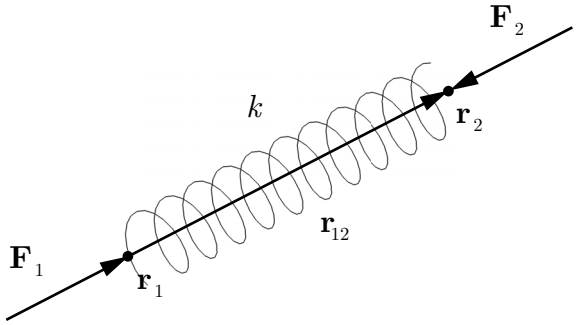


Figure 3.7: Air spring model.

The stiffness characteristics of the air springs have been estimated on the basis of the roll stiffness of the entire vehicle, k_φ, provided that, in this kind of vehi-

cles, around 60% of the roll stiffness is due to the air springs, and the remaining 40% is due to the anti-roll bars. The global roll stiffness coefficient can be estimated from catalogues, experience or similar vehicles, and is hereby considered as constant. Thus, the relationship between roll force and roll angle, φ, is linear. Also, roll stiffness can be divided into front ($k_{\varphi f}$) and rear ($k_{\varphi r}$) axle roll stiffnesses, associating them to the front and rear roll centers, respectively:

$$f_{\varphi f} = k_{\varphi f}\varphi \tag{3.2}$$

$$f_{\varphi r} = k_{\varphi r}\varphi \tag{3.3}$$

where the bodywork roll angle, φ, is measured around the roll axis. See Appendix B for the computation of the roll center and the roll axis of the coach.

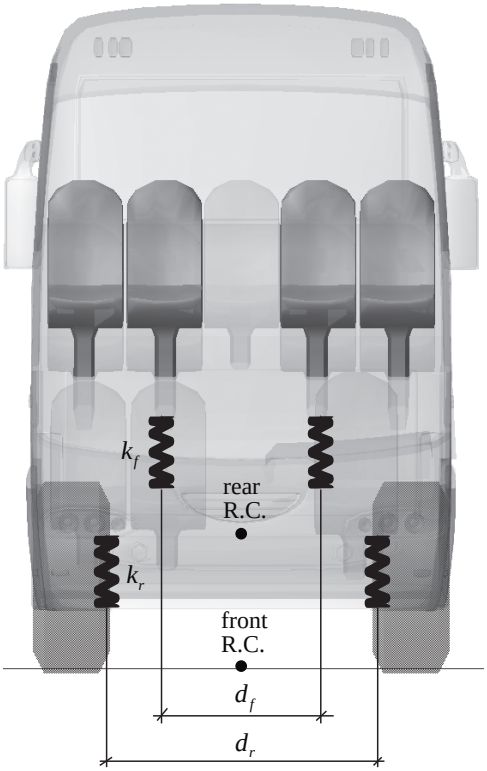


Figure 3.8: Front view of the springs.

Figure 3.8 shows a schematic representation of the air springs in a front view. Let k_f and k_r be the front and rear stiffness coefficients of the air springs, respectively. The equilibrium of torques in the front suspension can be written as:

$$0.6\ k_{\varphi f}\varphi = 2k_f\left(\frac{d_f}{2}\sin\varphi\right)\frac{d_f}{2} \tag{3.4}$$

Assuming that the roll angle is small and therefore $\sin\varphi \approx \varphi$, the relationship between the front roll stiffness and the front air spring stiffness can be calculated as follows:

$$0.6\ k_{\varphi f} = \frac{1}{2} k_f d_f^2 \tag{3.5}$$

The same reasoning can be applied to the rear suspension system:

$$0.6\ k_{\varphi r} = \frac{1}{2} k_r d_r^2 \tag{3.6}$$

Then, substituting the value of empirical values of the roll stiffness per axle ($k_{\varphi f} = 5.4 \times 10^5$ N·m/rad and $k_{\varphi r} = 5 \times 10^5$ N·m/rad), the air spring stiffnesses can be computed. See Table 3.4 for the specific values.

Coefficient	Value	Units
Front (k_f)	555.5	kN/m
Rear (k_r)	139.6	kN/m

Table 3.2: Air spring coefficients.

Dampers Quite similarly, the mathematical model of the dampers consists of a user force acting on both end points along the direction of the straight line that links them. This time, the magnitude is related to the projection of the relative velocity between the end points on the damper direction. Analogously, the damping forces are applied on the COG of the linked bodies with their corresponding torques. The expression of the damping force is:

$$\mathbf{F}_{d1} = -\mathbf{F}_{d2} = c(\mathbf{v}_{12}^T \mathbf{u})\mathbf{u} = c\left(\mathbf{v}_{12}^T \frac{\mathbf{r}_{12}}{\|\mathbf{r}_{12}\|}\right)\frac{\mathbf{r}_{12}}{\|\mathbf{r}_{12}\|} \tag{3.7}$$

where $\mathbf{F}_{di}$ is the damping force acting on point i of the body to which the damper is attached; $\mathbf{r}_i$ is the current Cartesian position of point i; $\mathbf{v}_i$ is the current velocity of point i; and c is the damping coefficient.

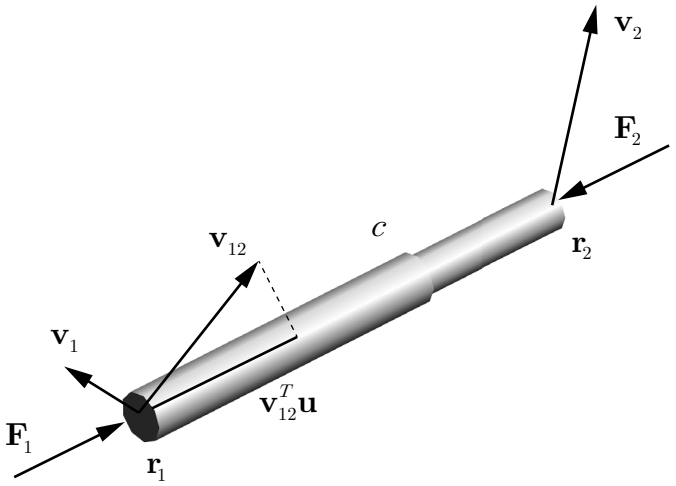


Figure 3.9: Damper model.

A trade-off between accuracy and computational efficiency is sought when regarding the damping coefficient value. According to Luque and Mántaras, 2003, a piecewise linear model is enough to capture the fundamental behavior of the damper. Figure 3.10 shows the four different slopes (c_i, $i = 1,...,4$) taken into account, depending on the relative velocity between the ends of the damper. The X-axis represents the relative velocity projected on the damper direction, $v = \mathbf{v}_{12}^T \mathbf{u}$. Parameters v_{11} and v_{22} establish the limit between linear regions. Positive relative velocities are the ones corresponding to damper rebound, and negative ones to compression.

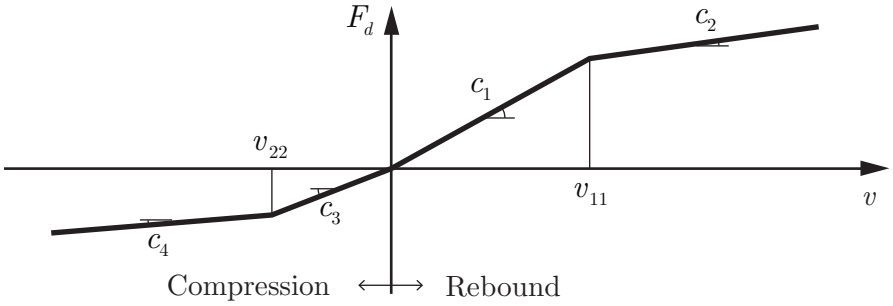


Figure 3.10: Damping coefficients of the damper model.

The equivalent analytical expression of the damping force would be:

$$F_d = \begin{cases} c_1 v_{11} + c_2(v - v_{11}) & \text{if } v_{11} \leq v \\ c_1 v & \text{if } 0 \leq v < v_{11} \\ c_3 v & \text{if } v_{11} < v < 0 \\ c_3 v_{22} + c_4(v - v_{22}) & \text{if } v \leq v_{22} \end{cases} \tag{3.8}$$

In the absence of data from the manufacturer, the shock absorber coefficients are supposed to have a value similar to those of analogous coaches (see Table 3.3). Note that these values are just a predesign and will be modified later by the optimization routine.

Coefficient	Value	Units
c_1	19.83	kN·s/m
c_2	9.915	kN·s/m
c_3	9.915	kN·s/m
c_4	4.957	kN·s/m
v_{11}	0.01	m/s
v_{22}	−0.01	m/s

Table 3.3: Damper coefficients.

Anti-roll bars Both anti-roll bars are analogous, but only the rear one is considered here as an example. The longitudinal parts of the anti-roll bar are considered as separate rigid bodies attached to the rear support by means of

revolute joints (see Figure 3.11). The transverse part is not considered, and is depicted only for aesthetic purposes. Both bending and torsional effects are merged into a single stiffness coefficient. The angle difference between the two end bodies is monitored, and a proportional torque is applied on the revolute joints. The force expression would be:

$$\|\mathbf{N}_1\| = \|\mathbf{N}_2\| = k_\theta(\theta_2 - \theta_1) \tag{3.9}$$

where $\mathbf{N}_1$ and $\mathbf{N}_2$ are the equivalent torques, θ_1 and θ_2 are the angles of the ends relative to an absolute reference, and k_θ is the angular stiffness coefficient.

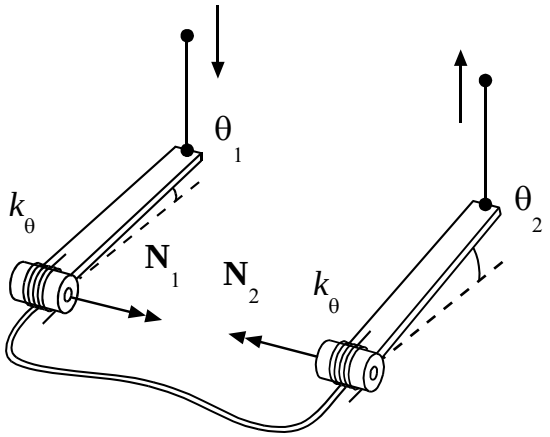


Figure 3.11: Anti-roll bar model.

According to experience with similar buses, around 40% of the roll stiffness is usually due to anti-roll bars. Since the global roll stiffness is known, the value of the stiffness coefficient k_θ can be estimated. The values of the anti-roll stiffness coefficients can be found in Table 3.4.

Coefficient	Value	Units
Front ($k_{\theta,f}$)	216	kN·m/rad
Rear ($k_{\theta,r}$)	200	kN·m/rad

Table 3.4: Anti-roll bar coefficients.

3.1.2 Steering system

In this model, the steering coordinate δ corresponds to the rotation of the steering actuator, which acts on the steering rods through the steering mechanism. A schematic 3D representation of the steering system can be seen in Figure 3.12.

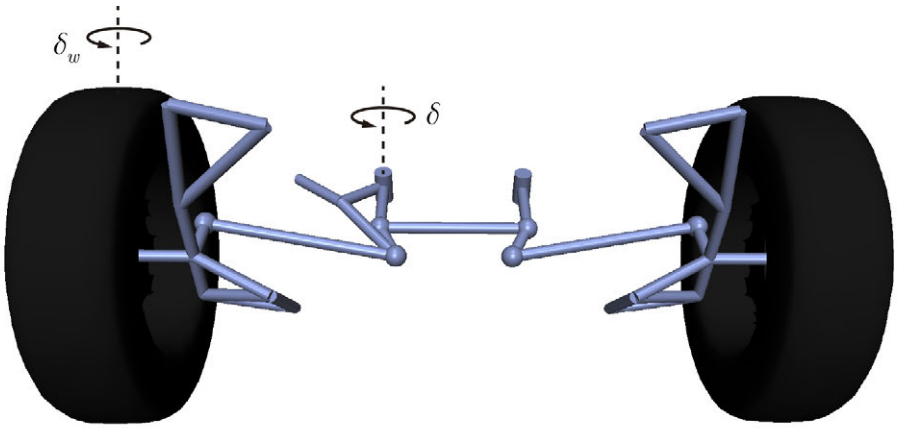


Figure 3.12: Steering system coordinates.

It is often useful to measure the wheel spin as well, as will become clear in the following sections. To that end, δ_w is defined as the rotation of the wheel with respect to the Z-axis. Both angles are related by a near-constant ratio $\delta \,/\, \delta_w$. In order to compute this relationship, a kinematic simulation is run. The steering coordinate is driven and the front wheel spins are measured and averaged out. The results of this simulation are shown in Figure 3.13. As can be seen, the relationship between both angles is pretty much constant. An average value of $\delta \,/\, \delta_w = 0.4615$ will be considered when needed in the following sections.

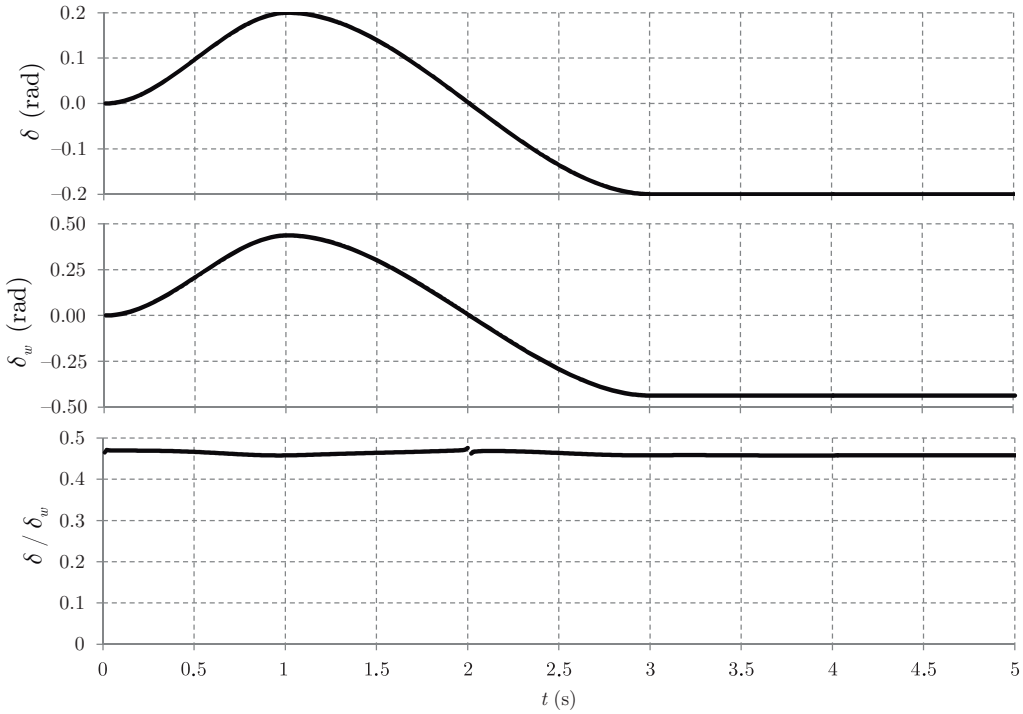


Figure 3.13: Relationship between steering coordinate and spin of front wheels.

In dynamic simulations, coordinate δ must be kinematically driven, and thus set by the user (usually as a function of time) in every step of the simulation. There are three basic strategies to do so:

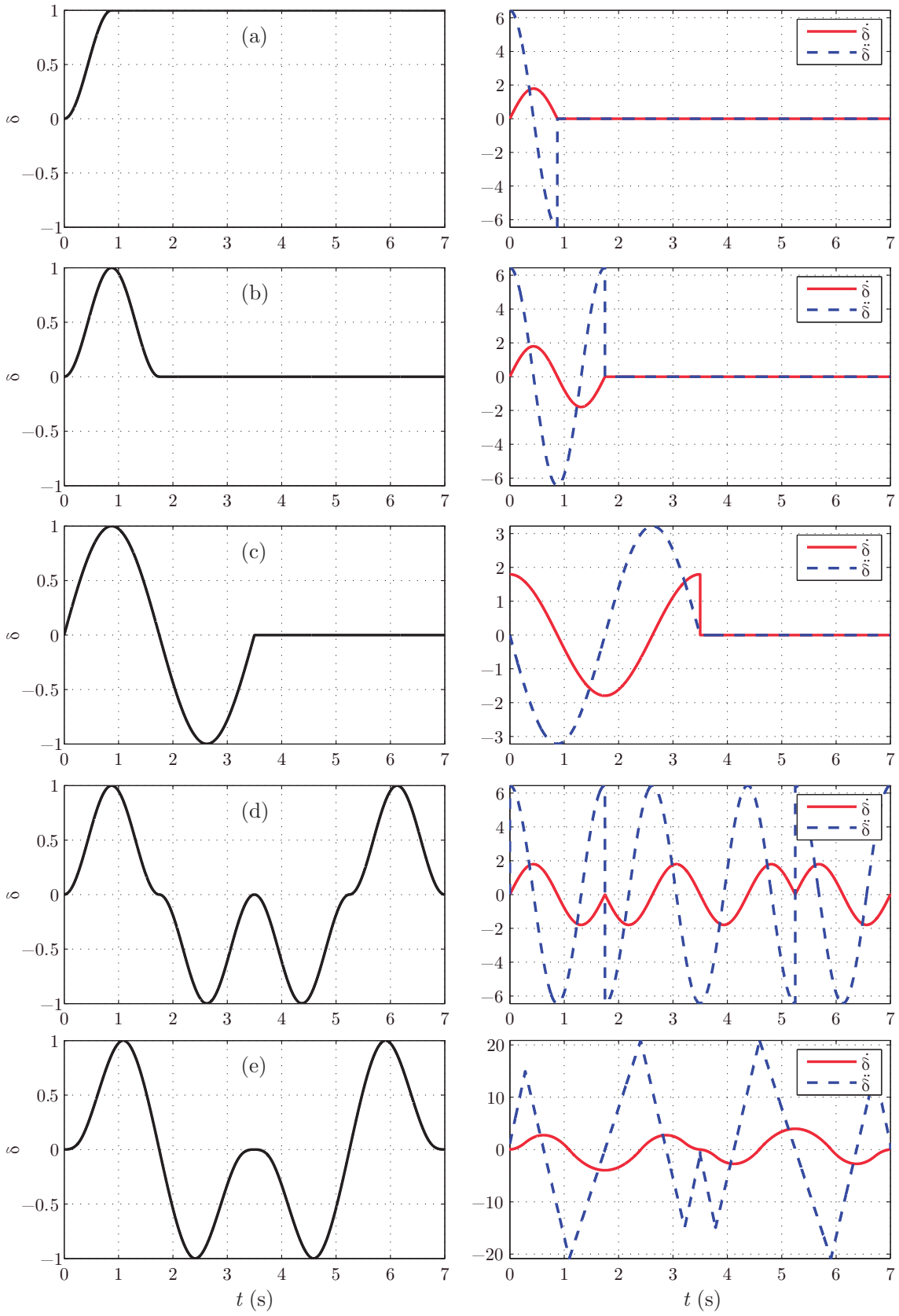


Figure 3.14: Sample unit steering functions and their first and second derivatives.

– Open-loop mathematical function. It does not take into account the effect of the steering value on the vehicle behavior, and is the most straightforward one. The steering value only depends on time.
– Human-in-the-loop (HITL) driver. The user operates a USB steering wheel whose commands are sent to the simulation program in real time, which allows the user to watch the vehicle behavior on the 3D graphical output. Interactive efficiency is paramount in this type of simulation.
– Closed-loop virtual driver. Includes the feedback of the vehicle behavior so as to minimize the error with respect to a predefined trajectory. This requires a controller to enforce the speed and the trajectory.

The first strategy is discussed in this chapter. The second one, even though implemented as well throughout this Thesis, is not presented here for the sake of brevity. Predefined steering functions are very useful in many applications where the trajectory and the final position of the vehicle are not essential. Some examples of predefined steering functions are gathered in Figure 3.14, specifically, a constant steering angle (a), a turn (b), a single lane-change (c) and a double lane-change (d, e).

3.1.3 Bodywork properties

The coach is considered unloaded for the initial setup and the static equilibrium stages, which means no passengers are seated on the seats. Allegedly, the COG of the unloaded bodywork is centered in the Y-direction. For X- and Z-positions, the global COG information is used. The global COG was measured in the X-direction using weighing scales, and in the Z-direction using a rollover test. Knowing the positions (from visual inspection) and weights (from similar vehicles) of the front and rear axles, the axles' COG can be subtracted from the global COG to obtain the bodywork's COG.

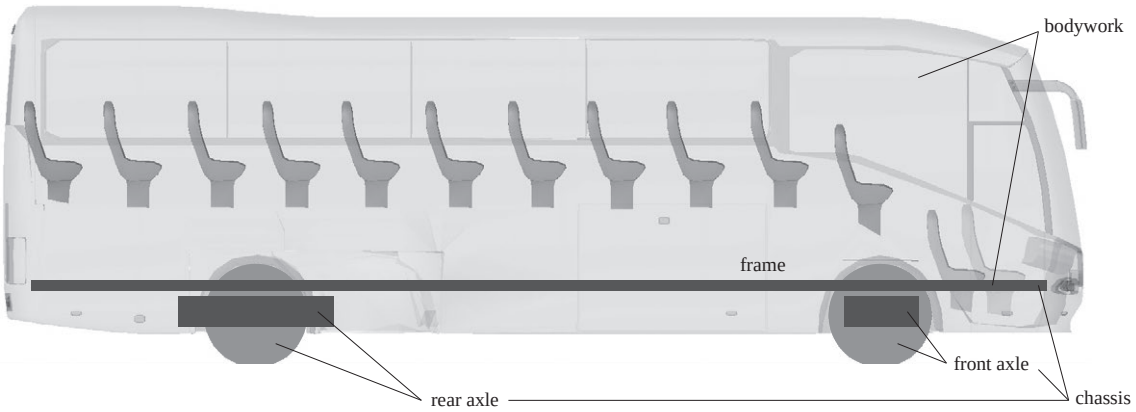


Figure 3.15: Parts of the coach.

The inertia tensor of the bodywork could not be computed using a realistic CAD model (including the frame, glass windows, seats, etc.) because such model

was not available, and exceeded the scope of this Thesis. It has therefore been estimated using the measured mass of the bodywork and the outer dimensions of the bodywork.

The flexibility of the chassis has to be considered for an accurate simulation of vehicle dynamics, according to Ambrósio and Gonçalves, 1999, and other authors. However, flexible multibody systems are out of the scope of this work as well, and thus a simpler approach is followed. The torsion stiffness of the bodywork is considered by dividing the bodywork into two separate bodies linked by a revolute joint with a torsion spring acting along the X-direction. This involves defining three different aspects: the position of the revolute joint, the stiffness of the angular spring and the distribution of inertia properties. The COG of the bodywork has been chosen as the position of the torsion joint, since it divides the mass of the bodywork into two in all directions. Thus, the bodywork is split by a vertical YZ-plane across that point.

The torsion stiffness coefficient, k_{chassis}, could be measured through a (probably destructive) torsion test with a real bodywork, or through a detailed FEM model of the assembly including glass windows, superstructure, etc. None of those could be used in the course of this Thesis, and therefore it had to be assimilated to the torsion stiffness of similar bodyworks ($k_{\text{chassis}} = 3 \times 10^5$ N·m).

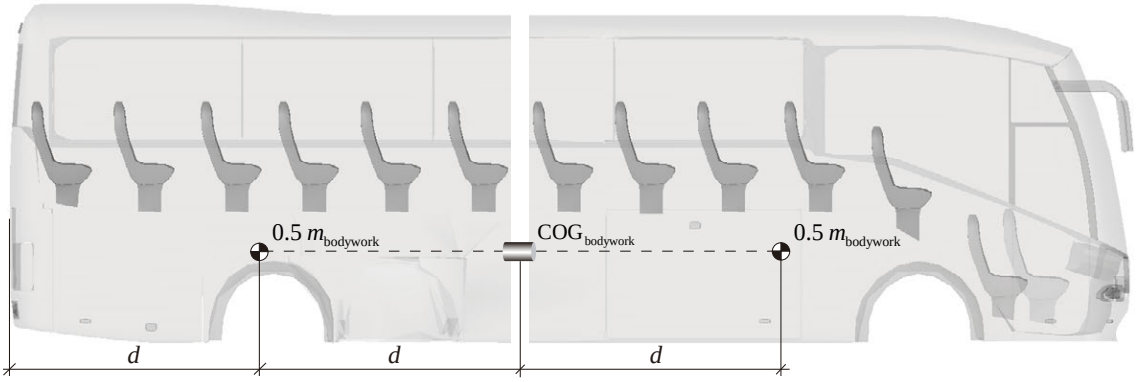


Figure 3.16: Division of the bodywork into two bodies.

Finally, the inertia properties have to be distributed. The position of the COG of each of the two bodywork halves is set so that the static moment of the weight forces in the COG of the original (undivided) bodywork is null. The inertia matrices are then computed with respect to the new COGs according to the size of the new bodywork parts.

The effect of the bodywork torsional stiffness on vehicle dynamics is of great importance, as has been widely proven in the literature. This influence is especially important when the bodywork (or chassis) is large, as is the case under study. To illustrate this issue, two simulations have been run: one with a flexible bodywork and the other one with a rigid bodywork. The maneuver is a 4-second double lane-change made up of splines (as shown in Figure 3.14) with 1-

millisecond time-step. The bodywork pitch and the COG vertical position have been plotted in Figure 3.17, showing that the cornering behavior is influenced by the bodywork flexibility. Specifically, the pitch is in general more acute when the bodywork is flexible, as is the COG vertical position.

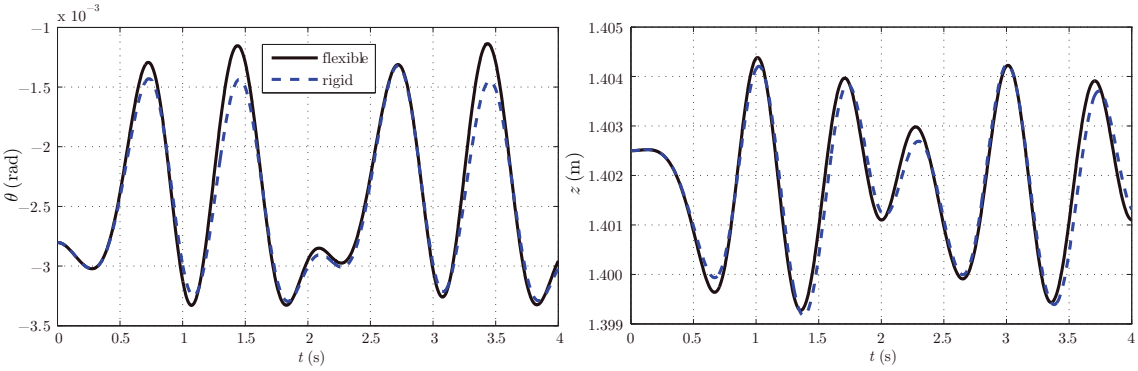


Figure 3.17: Effect of bodywork torsional flexibility on pitch and COG vertical position during a 4-second double lane-change maneuver.

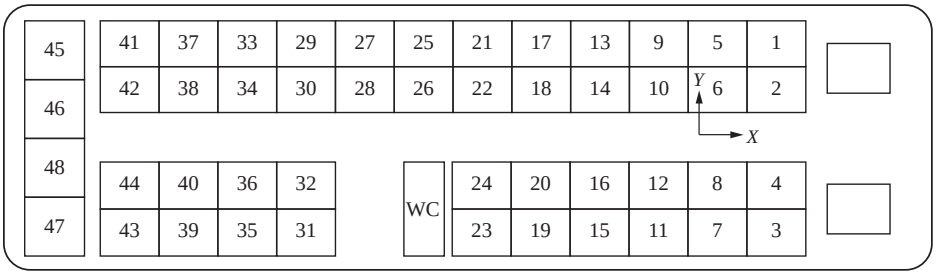


Figure 3.18: Passenger layout.

Finally, the passenger load is considered in order to capture the dynamic behavior of the coach in the simulated maneuvers more accurately. To this end, according to European Parliament Directives 97/27/EC and 2001/85/EC, a punctual mass of 71 kg is placed on each seat, including the driver and co-driver seats, which makes 3,550 kg in total. The layout of the bus seats is shown in Figure 3.18.

3.1.4 Multibody system topology

A complete enumeration of the joints between the model bodies is shown in Table 3.5. The identification of joints has been made through visual inspection. The flexibility of real bushings is not considered in this Thesis because they would require a completely different formulation (for that matter, see Hidalgo, 2013, and Ambrósio and Verissimo, 2009).

It is worth explaining how joints with more than one DOF are modeled. As explained in Section 2.2.1, the composition of the basic 1-DOF revolute (R) and prismatic (P) joints allow the user to model multi-DOF joints like spherical joints, universal joints, cylindrical joints, etc. To that end, massless auxiliary

bodies are introduced in between the two bodies linked by the compound joint. As an example, a spherical joint is considered. In this case, three rotational DOFs with two auxiliary (massless) bodies between them have to be superimposed. Each auxiliary body is defined by means of one point and two orthogonal unit vectors (see Figure 3.19). Thus, four points coincide in space (two from the auxiliary bodies and two from the real bodies). Bodies 1 and 2 are referred to as B1 and B2, and auxiliary elements 1 and 2 as A1 and A2, respectively. The joint DOFs are as follows:

– Revolute joint 1 between B1 and A1 in Z-direction.
– Revolute joint 2 between A1 and A2 in Y-direction.
– Revolute joint 3 between A2 and B2 in X-direction.

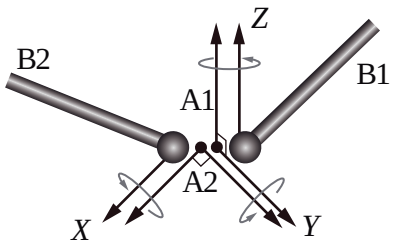


Figure 3.19: Compound spherical joint.

Another typical case is the 6-DOF joint (i.e., no joint at all) between the ground and the bodywork, and between the ground and the rear support. In this case, five auxiliary elements have to be added so as to define three revolute joints and three translational joints in the corresponding orthogonal directions.

On the other hand, the multibody formulation described in this Thesis considers the closed-loop multibody system as an open-chain system with tree structure, subject to closure-of-the-loop constraint equations, as has been thoroughly explained in Section 2.2.2. Three different closure-of-the-loop constraints have been implemented in the simulation program: the rod constraint, the spherical joint constraint and the revolute joint constraint. Certain rods, spherical joints and revolute joints must be removed from the closed-loop system and enforced later on through constraint equations. In practice, all rods have been considered as closure-of-the-loop constraints in the implementation, and the minimum number of spherical and revolute constraints (for the system to be a tree-structured system) have been defined.

#		Body i	Body k	Type	Direction (m)
1	Rear	FIXED	CH_AUX1	P	$\mathbf{u}_z$
2		CH_AUX1	CH_AUX2	P	$\mathbf{u}_y$
3		CH_AUX2	CH_AUX3	P	$\mathbf{u}_x$
4		CH_AUX3	CH_AUX4	R	$\mathbf{u}_z$
5		CH_AUX4	CH_AUX5	R	$\mathbf{u}_y$
6		CH_AUX5	R_CHASSIS	R	$\mathbf{u}_x$
7		FIXED	RS_AUX1	P	$\mathbf{u}_z$
8		RS_AUX1	RS_AUX2	P	$\mathbf{u}_y$
8		RS_AUX2	RS_AUX3	P	$\mathbf{u}_x$
10		RS_AUX3	RS_AUX4	R	$\mathbf{u}_z$
11		RS_AUX4	RS_AUX5	R	$\mathbf{u}_y$
12		RS_AUX5	R_SUPPORT	R	$\mathbf{u}_x$
13		R_SUPPORT	R_L_WHEEL	R	$\mathbf{u}_y$
14		R_SUPPORT	R_L_WHEEL_2	R	$\mathbf{u}_y$
15		R_SUPPORT	R_R_WHEEL	R	$\mathbf{u}_y$
16		R_SUPPORT	R_R_WHEEL_2	R	$\mathbf{u}_y$
17		R_SUPPORT	R_L_STAB	R	$\mathbf{u}_y$
18		R_SUPPORT	R_R_STAB	R	$\mathbf{u}_y$
19	Front	L_CARRIER	F_L_WHEEL	R	$\mathbf{u}_y$
20		R_CARRIER	F_R_WHEEL	R	$\mathbf{u}_y$
21		L_S_TRIANG	F_CHASSIS	R	$10^{-3}\{-1000, 8.6, 3.3\}^T$
22		L_I_TRIANG	F_CHASSIS	R	$-\mathbf{u}_x$
23		R_S_TRIANG	F_CHASSIS	R	$10^{-3}\{-1000, -8.6, 3.3\}^T$
24		R_I_TRIANG	F_CHASSIS	R	$-\mathbf{u}_x$
25		L_STEER	F_CHASSIS	R	$\mathbf{u}_z$
26		R_STEER	F_CHASSIS	R	$\mathbf{u}_z$
27		F_L_STAB	F_CHASSIS	R	$\mathbf{u}_y$
28		F_R_STAB	F_CHASSIS	R	$\mathbf{u}_y$
29		L_S_TRIANG	LST_AUX1	R	$\mathbf{u}_x$
30		LST_AUX1	LST_AUX2	R	$\mathbf{u}_y$
31		LST_AUX2	L_CARRIER	R	$\mathbf{u}_z$
32		L_CARRIER	L_I_TRIANG	S	-
33		R_S_TRIANG	RST_AUX1	R	$\mathbf{u}_x$
34		RST_AUX1	RST_AUX2	R	$\mathbf{u}_y$
35		RST_AUX2	R_CARRIER	R	$\mathbf{u}_z$
37		R_CARRIER	R_I_TRIANG	S	-
38		R_CHASSIS	F_CHASSIS	R	$\mathbf{u}_x$

Table 3.5: List of the model joints.

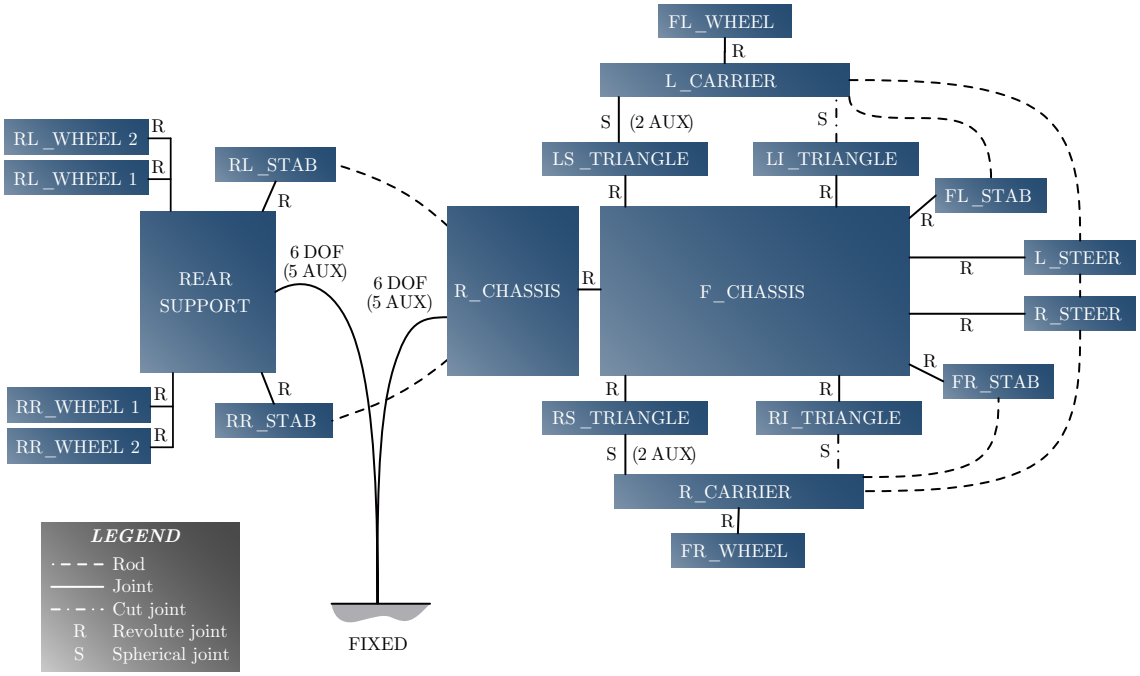


Figure 3.20: Coach model topology.

Due to the fact that the rear support is linked to the chassis only by rods (which are removed to obtain an open-chain tree), two branches start from the fixed element: one goes to the rear support, from which the wheels and other parts branch, and the other to the rear part of the bodywork, from which the triangles and other parts stem from. The two subsystems bodywork-triangle-upright-triangle-bodywork make up two closed-loop systems which need to be converted into open-loop subsystems. To that end, the two lower spherical triangle-upright joints are transformed into closure-of-the-loop constraints. Therefore there are 2 cut spherical joints and 11 rods. The topological structure of the system has been depicted in Figure 3.20, showing the links between the bodies and the types of joints. FL means “front left”, RR “rear right”, and so on.

3.1.5 External forces

A real vehicle is subject to many kinds of external forces, but only the most relevant ones are considered here. For instance, the aerodynamic lift and drag forces are neglected. The terrain is supposed to be flat except for ride comfort simulations (see Section 3.3), where a noise profile will be applied to the wheels. Traction is directly applied as a torque on the motor wheels, which is controlled in terms of a prescribed forward velocity. Thus, the transmission is not modeled. The traction torque is applied on the revolute joints between each of the rear wheels and the rear support, being the control law:

$$T(t) = 10^4 T_n \left| v_n - v(t) \right| \left[v_n - v(t) \right] \tag{3.10}$$

where $T_n = 600\ \mathrm{N \cdot m}$ is the vehicle-dependent nominal torque, v_n is the nominal vehicle velocity, $v \equiv \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}$ is the current vehicle velocity, and x, y, and x are the Cartesian coordinates of the bodywork COG. Because of the rolling resistance, $\left[v_n - v(t)\right]$ is always going to be positive when riding along flat roads, and thus no braking is necessary.

Tire forces are crucial for vehicle dynamics, as they provide the link between the vehicle and the ground. The vehicle speeds up, slows down and negotiates turns through them. The physical phenomena taking place in the tires are very complicated, and the theoretical models available in the literature always assume simplifications. There are at least four types of tire models: simple models, empirical models, physical models and finite-element models. For frequencies up to 8 Hz, one of the most accurate and widespread models is Pacejka's Magic Formula (see Pacejka, 1996, and 2002). It consists of a set of semi-empiric equations that embody the dynamical behavior of tires with reasonable accuracy.

The coach has six 295/80 R22.5 wheels. A simplified version of the 2002 Magic Formula (see Pacejka, 2002) has been used throughout this Thesis. Only quasi-static (not transient) effects are considered, because most dynamic maneuvers stay within the range of tire linearity, i.e., lateral accelerations will stay under 0.4 g for the most part.

For vertical (radial) behavior, Pacejka's tire model considers the tire as a linear spring-damper set between the wheel center and the contact point C. The contact point is computed assuming that the wheel is a rigid disk (see Figure 3.21). From a unit vector orthogonal to the wheel plane ($\mathbf{u}_A$) and a vertical unit vector $\mathbf{u}_n$ (orthogonal to the ground), we can easily compute a tangent unit vector $\mathbf{u}_t$ and a radial unit vector $\mathbf{u}_r$. The position of the contact point would then have the expression:

$$\mathbf{r}_C = \mathbf{r}_A + r\,\mathbf{u}_r \tag{3.11}$$

from which the value of the deformed radius r can be found assuming that the contact point $\mathbf{r}_C$ belongs to the XY-plane.

After the contact point is computed, the Magic Formula provides expressions for the longitudinal force (F_x) and side forces (F_y and M_z). The former depends explicitly on the longitudinal slip, while the latter depend on the side slip. In both cases, the dependency with respect to the vertical load F_z and the camber angle γ is implicitly accounted for.

It is well known that the longitudinal velocity of the vehicle does not match any of the velocities computed from the angular velocity of the tires. The reason for this, besides the loss of grip, is the elastic deformation of the tires due to the contact with the ground. A diagram of the different velocities present in the wheel is depicted in Figure 3.22.

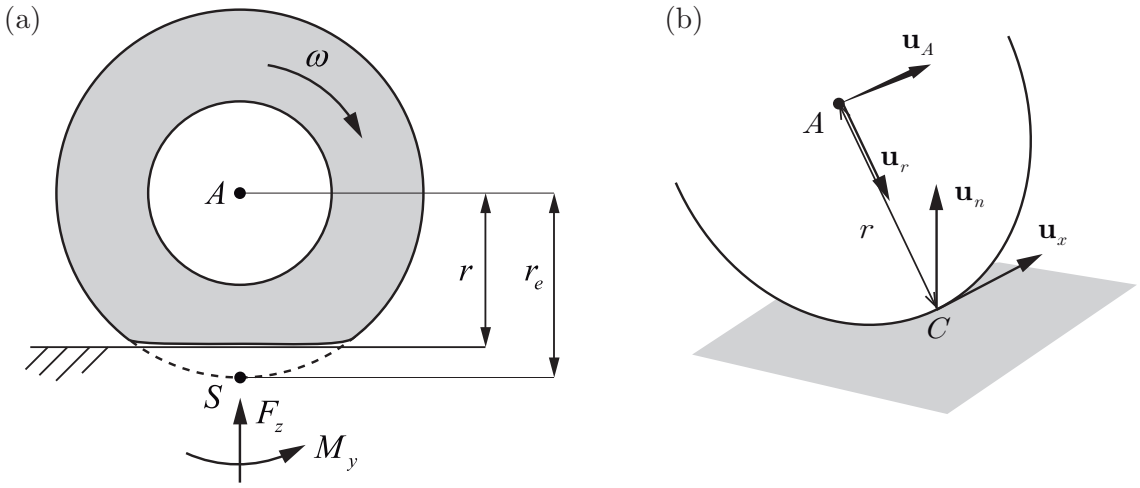


Figure 3.21: (a) Deformed wheel. (b) Contact point.

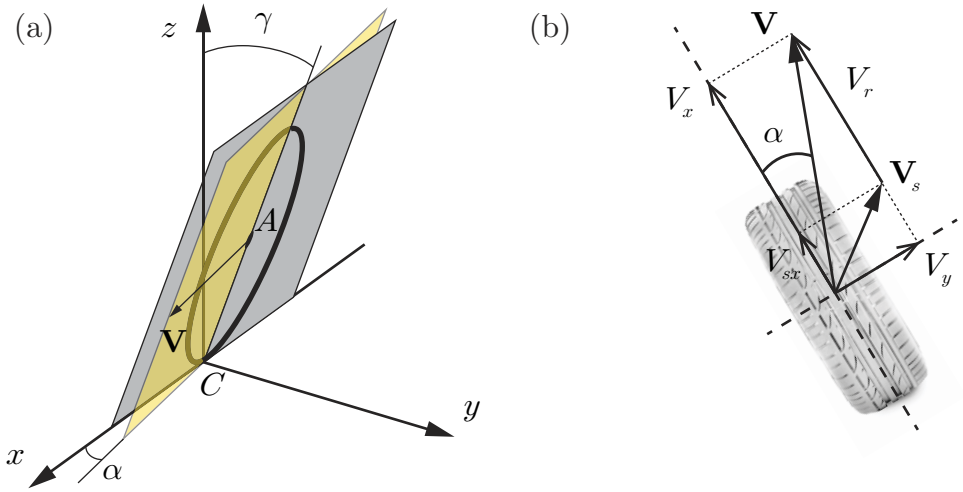


Figure 3.22: Wheel velocities.

where $\mathbf{V}$ is the velocity of the wheel center, $\mathbf{V}_s$ is the slip velocity and V_r is the rolling velocity. The longitudinal slip, κ, can be defined as the percentage difference between the wheel center longitudinal velocity and the rolling velocity of that same point computed from the effective (undeformed) radius:

$$\kappa \equiv -\frac{V_{sx}}{|V_x|} = -\frac{V_x - V_r}{|V_x|} = -\frac{V_x - \omega r_e}{|V_x|} = \frac{\omega r_e}{|V_x|} - 1 = \frac{r_e}{r} - 1 \tag{3.12}$$

where r is the loaded (deformed) radius and r_e is the effective rolling radius (i.e., the radius of the wheel purely rolling in a flat surface under null brake and driving torques). Regarding lateral behavior, it is important to remark that the orientation of the wheels does not match the angle of the vehicle velocity when taking a turn. The side slip angle, α, is the angle between the velocity of the wheel center and the wheel plane:

$$\tan\alpha \equiv -\frac{V_{sy}}{|V_x|} \tag{3.13}$$

Note that when V_x is nearly zero, a small quantity ε must be added to the denominator of slip expressions. Pacejka's tire model considers three different cases based on the typical slip situations:

- Pure longitudinal slip: rolling in a straight line with a motor or brake torque.
- Pure lateral slip: taking a turn with no torque applied.
- Combined effect: simultaneous longitudinal and lateral slips.

According to these cases, different formulas are defined. In the two pure-slip cases, longitudinal force F_x and side force F_y are modeled with the same type of expression, which can be written as:

$$\begin{aligned} y &= D \sin\Big\{C \arctan\big[Bx + EBx - E \arctan Bx\big]\Big\} \equiv Y - S_V \\ x &\equiv X + S_H \end{aligned} \tag{3.14}$$

where $Y = S_V + y$ is the variable under study (F_x or F_y) and X is the corresponding longitudinal slip ratio (κ) or side slip angle tangent ($\tan\alpha$). Naturally, longitudinal magnitudes are functions of the slip ratio and lateral magnitudes are functions of the slip angle tangent. On the other hand, the meaning of the coefficients present in the equation is:

- B is the stiffness factor, which controls the slope at the origin ($x = y = 0$).
- C is the shape factor.
- D is the peak value (at $x = x_m$).
- E is the curvature factor, which controls the curvature and horizontal position of the peak.
- BCD is the slope at the origin ($x = y = 0$).
- S_H and S_V are the curve offsets with respect to the origin (see Figure 3.23), accounting for the rolling friction and the wheel camber angle.

These coefficients are case-dependent, and can be interpolated from table values obtained from tire tests or calculated through formulas. The tire coefficients used in the coach model are gathered in Appendix C.

Self-aligning torque, M_z, is caused by the displacement of force F_y with respect to the contact point. Such displacement has the shape of a cosine function. Thus, the equation for the self-aligning torque is:

$$y = D \cos\Big\{C \arctan\big[Bx + EBx - E \arctan Bx\big]\Big\} \tag{3.15}$$

where x is again the lateral slip ($\tan\alpha$).

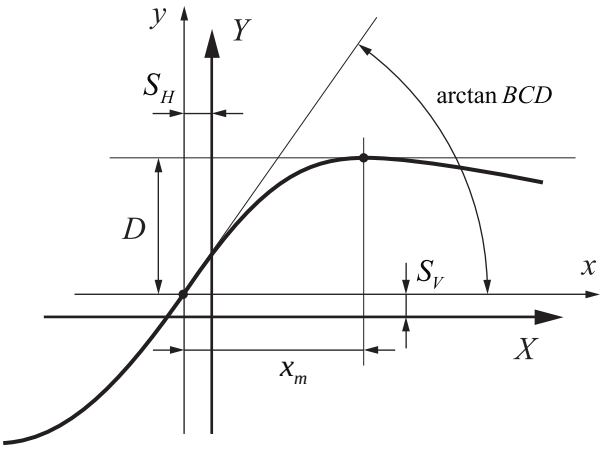


Figure 3.23: Generic tire curve.

The combined effect of longitudinal and slide slips modifies the curves with further coefficients and accounts for additional forces like the normal load, F_z, the overturning couple, M_x, and the rolling resistance moment, M_y. These further terms are not detailed here, but can be found at Pacejka, 2002.

In the case of the dual wheels at the rear axle, each wheel has its own set of tire forces and can roll independently. This approach accounts for the different lateral position of the wheel centers and the different rolling velocities.

MD Adams	Event / Maneuver	ADAMS/ Handling Tire							Specific Models	
		PAC2002[1]	PAC-TIME[1]	PAC89[1]	PAC94[1]	FIALA[1]	5.2.1.[1]	UA Tire[1]	PAC-MC[1]	FTire
Handling	Stand still and start	+	o/+	o/+	o/+	o/+	o/+	o/+	o/+	+
	Parking (standing steering effort)	+	-	-	-	-	-	-	-	+
	Standing on tilt table	+	+	+	+	+	+	+	+	+
	Steady state cornering	+	+	o/+	+	o	o	o/+	+	o/+
	Lane change	+	+	o/+	+	o	o	o/+	+	o/+
	ABS braking distance	+	o/+	o/+	o/+	o	o	o/+	o/+	+
	Braking/power-off in a turn	+	+	o	o	o	o	o	+	o/+
	Vehicle Roll-over	+	o	o	o	o	o	o	o	+
	On-line scaling tire properties	+	-	-	-	-	-	-	-	o
Ride	Cornering over uneven roads *	o/+	o	o	o	o	o	o	o	o/+
	Braking on uneven road *	o/+	o	o	o	o	o	o	o	+
	Crossing cleats / obstacles	-	-	-	-	-	-	-	-	+
	Driving over uneven road	-	-	-	-	-	-	-	-	+
	4 post rig (A/Ride)	+	o/+	o/+	o/+	o/+	o/+	o/+	o/+	o/+
Chassis Control	ABS braking control	o/+	o	o	o	o	o	o	o	+
	Shimmy[2]	o/+	o	o	o	o	o	o	o	+
	Steering system vibrations	o/+	o	o	o	o	o	o	o	+
	Real-time	+	-	-	-	-	-	-	-	-
	Chassis control systems > 8 Hz	o/+	-	-	-	-	-	-	-	+
	Chassis control with ride	-	-	-	-	-	-	-	-	+
Dura-bility	Driving over curb	-	-	-	-	-	o	o	-	o/+
	Driving over curb with rim impact	o	-	-	-	-	o	o	-	o/+
	Passing pothole	-	-	-	-	-	o	o	-	o/+
	Load cases	-	-	-	-	-	o	o	-	o/+

-	Not possible/Not realistic
o	Possible
o/+	Better
+	Best to use

* wavelength road obstacles > tire diameter
[1] use_mode on transient and combined slip
[2] wheel yawing vibration due to supension flexibility and tire dynamic response

Table 3.6: Tire models and their applicability.
Courtesy of MSC.Software Corporation[5].

[5] © MSC.Software Corporation. All rights reserved.

Finally, it is worth clarifying the range of application of Pacejka's model. As a way of illustration, Table 3.6 shows Adams recommendations for common tire models. The model presented here would correspond to the steady-state part of Adams' PAC2002 model. According to this table, the tire model should be used with caution in ride comfort, chassis control and durability applications.

3.1.6 Static equilibrium position

Models of complex mechanical systems are first modeled in a zero gravity environment, and often need a static equilibrium analysis prior to the dynamic analysis, in order to compute the steady state equilibrium position. In the case of vehicles, it is also advisable to perform such kind of analysis so that the vehicle and the suspension systems are in a "relaxed" state at the beginning of the simulation. The reason for this is that the initial configuration of the system is often not a static equilibrium one, due to:

– Differences between the model forces and the real forces
– Differences between the model geometry and the real geometry

Let us focus on the position of the bodywork. The initial (non-equilibrium) bodywork geometry was measured on the real vehicle using a coordinate-measuring machine. However, the subsystems between the bodywork and the ground (air springs, unidentified unsprung mass, tire stiffness, etc.) need to be configured appropriately to cope with the forces transmitted to the ground. In an ideal, perfectly set up model, the initial position of the vehicle would coincide with the static equilibrium configuration of the real vehicle. This is often not the case.

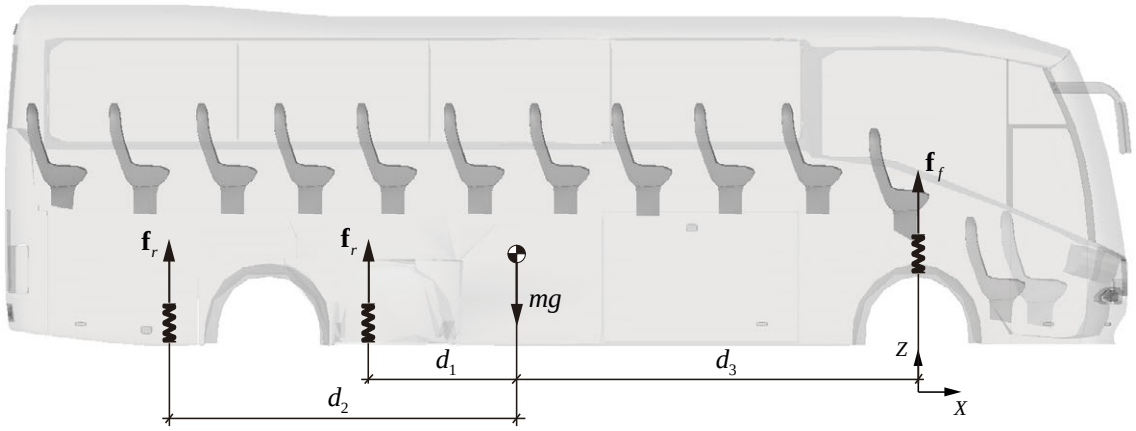


Figure 3.24: Diagram of the spring preloads.

Beyond the definition of inertia properties and the accurate definition of geometry, which have already been carried out, the preloading of air springs and initial compression of tires play an important role in the static equilibrium. The preload of the air springs has been estimated considering that they must with-

stand the weight of the bodywork. The values have then been adjusted so that the final state of the bodywork resembles the steady state of the real bodywork.

As a tough estimation of the air spring preloads, a 2D equilibrium of the bodywork can be posed (see Figure 3.24). In this situation, the anti-roll bars are inactive, since there is no bodywork roll, and therefore the bodywork weight is entirely borne by the air springs. Assuming that all springs exert their force on the bodywork at the same height, and that the four rear springs have the same preload, the values of the preload forces can be found from the scalar version of the equilibrium of moments and forces:

$$f_f = \frac{mg(d_1 + d_2)}{2d_3 + d_1 + d_2} \tag{3.16}$$

$$f_r = \frac{1}{2} mg \left(1 - \frac{d_1 + d_2}{2d_3 + d_1 + d_2} \right) \tag{3.17}$$

where f_f and f_r are the front and rear preload forces respectively, m is the mass of the bodywork, g is gravity, and d_1, d_2 and d_3 are the horizontal distances from the spring forces to the bodywork COG, as depicted in Figure 3.24. These estimated preloads still need to be adjusted heuristically so that the suspension design meets the requirement that the bodywork position at the end of the static equilibrium simulation is the one of the real vehicle.

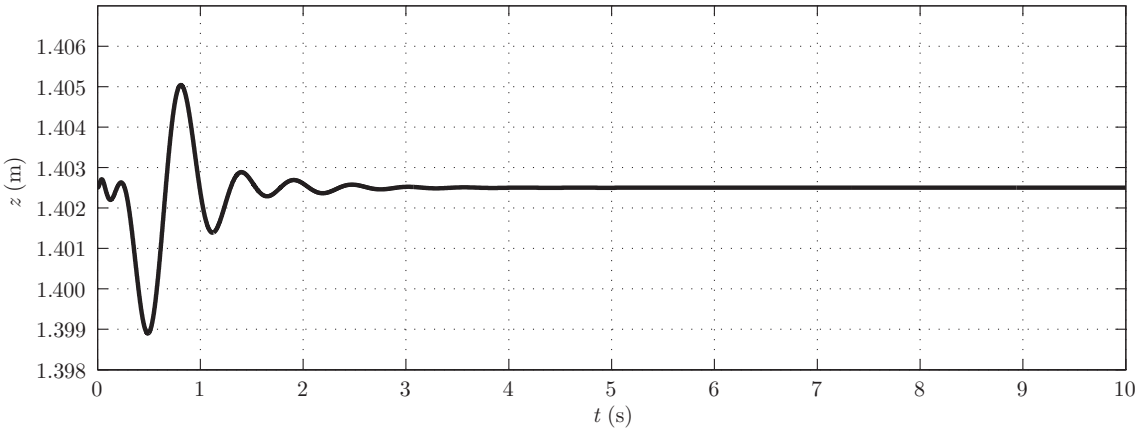


Figure 3.25: Vertical position of the bodywork during the static equilibrium analysis.

It goes without saying that the velocity of the vehicle in the static equilibrium position is null. Under these conditions, ground reactions cannot be computed univocally because the system is statically overdetermined. Provided there are six tire reactions (three translational forces and three torques) between each wheel and the ground, the number of unknowns is a lot larger than the number of equilibrium equations. For this reason, tire forces in the static equilibrium position are replaced by vertical springs with the stiffness of the tires. In this

regard, it is not clear whether commercial packages like Adams/Car or CarSim use the full tire model or a simplified one when the vehicle velocity is null.

To prevent longitudinal and side vehicle translations in the static equilibrium simulation, the X- and Y-coordinates and the roll (φ) and yaw (ψ) angles of the bodywork are blocked (i.e., kinematically driven to zero values). This eliminates small longitudinal and side translations coming from the small differences in the symmetry of the steering system and from unbalanced numerical errors. Thus, the only allowed DOFs of the bodywork are the Z-coordinate and the pitch (θ) angle. This technique helps to control the static equilibrium simulation, and has proven to be very effective.

After the adjustment of the spring preloads of the tires and the blockage of some bodywork DOFs, the static simulation is carried out. The system starts oscillating until the energy is dissipated on the dampers and the equilibrium is reached. Basically, the subsystems between the bodywork and the ground readapt to the equilibrium configuration as the global state of the bodywork reaches the desired value. Figure 3.25 shows the vertical displacement of the coach bodywork when the static equilibrium simulation is performed from the initial conditions. After the sprung mass and intermediate parts rearrange, the position of the bodywork reaches the real (measured) value.

3.2 Handling response

Good dynamic behavior is crucial for a road vehicle to ensure that it is safe. However, the evaluation of the dynamic response is hard because the driver and the vehicle constitute a closed-loop system which is difficult to simulate and with an inherently low degree of repeatability. Even if considered separately, driver and vehicle models are not easy to analyze. For instance, small (and difficult to control) variations in the road and tire conditions can greatly affect the measured results. Moreover, conclusions cannot always be generalized or extrapolated to other vehicles.

In general, the handling response of a vehicle can be defined as the ability to follow the driver's steering commands at any time under unfavorable conditions, with precision and promptness. Physically speaking, the ability of a vehicle to change direction is linked to the amount of lateral acceleration it can handle over time. Nevertheless, handling response is also related to the subjective experience of the driver and the feedback he or she gets from the vehicle. The ultimate goal of handling is to always grant the driver the control of the vehicle, which enables an accurate road holding.

There are several ways in which handling response can be assessed. Usually suspension designs are tested by performing real or virtual dynamic maneuvers

where meaningful magnitudes are monitored. Some examples of handling maneuvers are double lane-change (DLC) maneuvers, obstacle-avoidance maneuvers, step-steer (SS) maneuvers, constant speed (CS) maneuvers, severe braking and severe speed-up. Two examples of monitored magnitudes are the roll angle, φ, and the lateral acceleration, a_y.

3.2.1 Background

The most common handling defect is the lack of *directional stability*, which is deeply related to the tire grip. As explained before in Section 3.1.4, tire slip can be lateral or longitudinal. In general, tire forces F_x, F_y and M_z depend nonlinearly on the longitudinal slip κ, the side slip α, the camber angle γ and the normal force F_z (see Figure 3.22). When lateral loads appear, the side force grows with the side slip angle up to a point, and then it starts decreasing towards the loss of grip. With respect to the axle, depending on the values of the front and rear axle slips, the vehicle has an *oversteering* or an *understeering* behavior. The former happens when the front axle slip is larger than the rear one, and vice versa. Slip angles have a great influence on the amount of lateral force a tire can absorb. When an entire axle loses the ability to handle the lateral force, it loses grip and the driver loses control.

When the tire is subject to a combination of both longitudinal and lateral forces, it is useful to depict the slip limits using a friction ellipse, as seen in Figure 3.26. The ellipse delimits the area in which combinations of longitudinal and lateral forces are safe. The points located outside of the ellipse are considered unsafe. In this Thesis, though, the focus is placed on purely lateral stability, and thus the maneuvers will be carried out at constant speeds with small longitudinal tire forces.

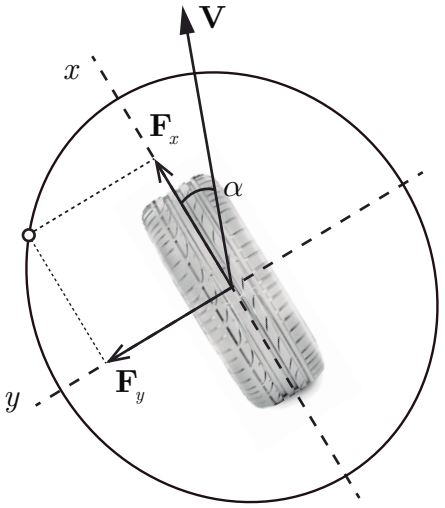


Figure 3.26: Tire friction ellipse.

However, lateral forces also depend on the normal force, which is why slip issues are aggravated by severe cornering maneuvers. When a vehicle is turning, centrifugal forces appear due to lateral acceleration, and the load is transferred from the inside of the vehicle to the outside. The higher the speed and the smaller the radius of the curve, the bigger the load transfer. A load transfer between both sides of the vehicle means that the outer wheels increase their normal force and the inner ones decrease it. For a specific axle i, load transfer $\Delta F_{z,i}$ is related to the ratio of the COG height and the track width:

$$\Delta F_{z,i} \propto \frac{\mathrm{COG}_{\mathrm{height}}}{\mathrm{track}_i} \tag{3.18}$$

A high load transfer can lead to an undesirable loss of grip. Based on Figure 3.27, let us explain why. Except for very large values of the normal force, the lateral force of a single tire grows nonlinearly with the normal force. However, the global behavior of an axle with two tires is not like that. During vehicle cornering, a load transfer takes place, thus reducing the normal and lateral forces of one tire and increasing the normal and lateral forces of the other. It turns out that, because of the nonlinear dependency of the lateral force with normal force, the global (average) side force is smaller than the case in which no load transfer was present. Thus, for the axle to sustain the same lateral force as without load transfer, an additional slip $\Delta\alpha$ must be spent. Tire manufacturers provide, for each tire and road friction conditions, the optimum normal force.

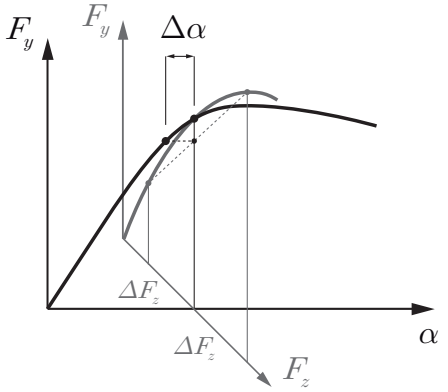


Figure 3.27: Influence of lateral load transfer on tire characteristics.

Lateral slip usually happens before vehicle rollover, and thus directional stability is often a bigger risk than *roll stability*. However, roll instability might happen in specific situations. For a vehicle to roll over, grip must be ensured by a high road friction or by a tripping mechanism such as a curb (see Martín, 2013). Rollover is triggered by the roll moment that appears when negotiating curves, which in turn is caused by large centrifugal forces or a high position of the COG. Lateral acceleration is therefore crucial for the analysis of roll stability, and the way it develops over time can be assessed as a vehicle handling charac-

teristic. There are several ways of computing lateral acceleration. Lateral (or normal) acceleration, a_y, can be defined as the component of the bodywork acceleration perpendicular to the vehicle velocity. According to this definition, the following expression can be written:

$$\mathbf{a}_y = \mathbf{a} - \mathbf{a}_x = \mathbf{a} - (\mathbf{a}^T \mathbf{u}_x)\mathbf{u}_x = \mathbf{a} - \left(\mathbf{a}^T \frac{\mathbf{v}}{\|\mathbf{v}\|}\right)\frac{\mathbf{v}}{\|\mathbf{v}\|} \tag{3.19}$$

where $\mathbf{a} \equiv \{\ddot{x}, \ddot{y}, \ddot{z}\}^T$ is the acceleration, $\mathbf{a}_y$ is the lateral (normal) acceleration vector, $\mathbf{a}_x$ is the longitudinal (tangent) acceleration vector, $\mathbf{v} \equiv \{\dot{x}, \dot{y}, \dot{z}\}^T$ is the velocity vector, $\mathbf{u}_x$ is the longitudinal (tangent) unit vector and x, y and z are the Cartesian coordinates of the chassis' COG. This way of expressing $a_y \equiv \|\mathbf{a}_y\|$, although completely general, would not be competitive when used to compute the sensitivities of the lateral acceleration in Chapter 5. Therefore, two approximate equations are proposed. When velocity and acceleration vectors ($\mathbf{v}$ and $\mathbf{a}$) are parallel to the XY-plane, it is easier to compute the normal unit vector, $\mathbf{u}_y$. This is true when the vehicle is riding along a flat track, which will be the case throughout this Thesis. According to this assumption and using a component-wise nomenclature:

$$a_y = \mathbf{a}^T \mathbf{u}_y = \begin{Bmatrix} \ddot{x} \\ \ddot{y} \\ 0 \end{Bmatrix}^T \begin{Bmatrix} \dot{y} \\ -\dot{x} \\ 0 \end{Bmatrix} \frac{1}{\sqrt{\dot{x}^2 + \dot{y}^2 + 0^2}} = (\ddot{x}\,\dot{y} - \dot{x}\,\ddot{y}) \frac{1}{\sqrt{\dot{x}^2 + \dot{y}^2}} \tag{3.20}$$

In addition to the assumption of the track being flat, when a longitudinal velocity control is implemented, as is the case here (see Section 3.1.5), acceleration is lateral for the most part. Thus, a further step can be taken and lateral acceleration can be computed efficiently as:

$$a_y \approx a = \sqrt{\ddot{x}^2 + \ddot{y}^2} \tag{3.21}$$

Computing lateral acceleration this way is a lot more efficient and simple than using Eqs. (3.19) and (3.20), especially when it comes to calculating its sensitivities (both numerically and automatically, in Chapter 5) and optimizing handling response (Chapter 6). The error in Eq. (3.21) is very small, provided a good longitudinal velocity control is implemented. For instance, when performing a 14-s double lane-change maneuver, Figure 3.28 shows the approximated value of the lateral acceleration (Eq. (3.21)) versus the value of the exact one (Eq. (3.20)), as well as the absolute error between them. It is clear that the error is fairly small (around 1% at most).

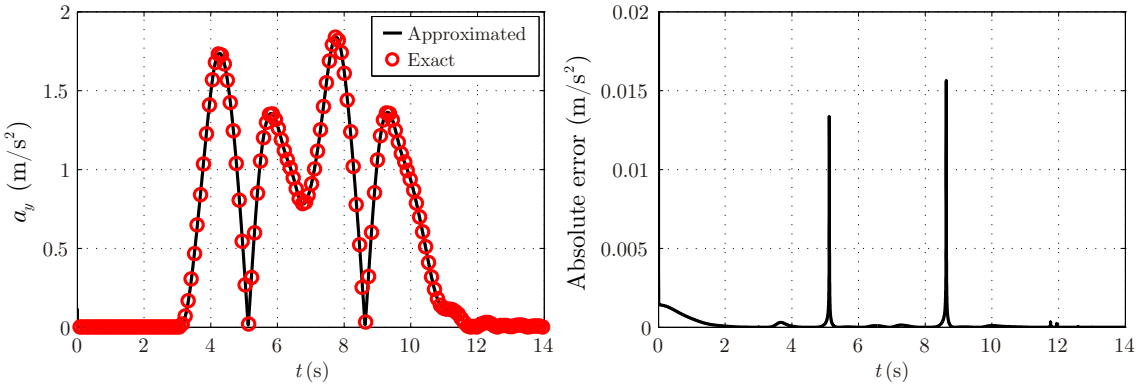


Figure 3.28: Approximated and exact lateral accelerations in a lane-change maneuver.

In terms of global vehicle characteristics, an approximation of the lateral acceleration that a vehicle can handle without rolling over is given by:

$$a_y \approx \frac{1}{2}\frac{\text{track}}{z_{\text{COG}}} \tag{3.22}$$

Note that, in a dynamic maneuver, rollover could also be caused by the excitation of the roll mode of vibration, by a sudden change of road friction or by the combined effect of braking and steering.

As far as lateral acceleration limits are concerned, 0.4 g is often considered as the limit for a safe vehicle response, according to Lechner, Ferrandez and Fleury, 1983. For accelerations greater than 4 g, only skilled drivers would be able to control the vehicle, and thus cannot be considered as a good handling experience. The differences in driving style can be easily depicted in what is called an acceleration ellipse, which is related to the friction ellipse.

Finally, anti-roll bars are often used to reduce vehicle roll. As explained in the modeling sections, the anti-roll bar acts between both ends of the axle when one end is lifted higher than the other one. Anti-roll bars can also be used to tune the handling balance of a car. If the roll stiffness of the front axle is increased, the front load transfer and the front side slip will increase too. This will stress the understeering behavior of the vehicle. Should the rear roll stiffness be increased, the opposite would happen and the oversteering behavior would be stressed. Additionally, active suspension systems help reduce vehicle roll at the cost of high energy consumption and manufacturing costs.

3.2.2 Lane-change maneuver

The severe *lane-change maneuver* is a classical test of the lateral stability of vehicles. It is sometimes called the *moose test*, as it consists of avoiding an obstacle in the road by turning first to the adjacent lane and then turning back into the original lane in an abrupt manner. Depending on the maneuver speed, the tire quality, the height of the vehicle COG and the suspension design, this

maneuver can be critical for the lateral stability. In extreme cases, the vehicle can roll over or lose grip, which is obviously not desirable. Thus, the suspension design must guarantee a good lane-change performance, being in fact one of the typical design requirements for the manufacturing of new vehicles.

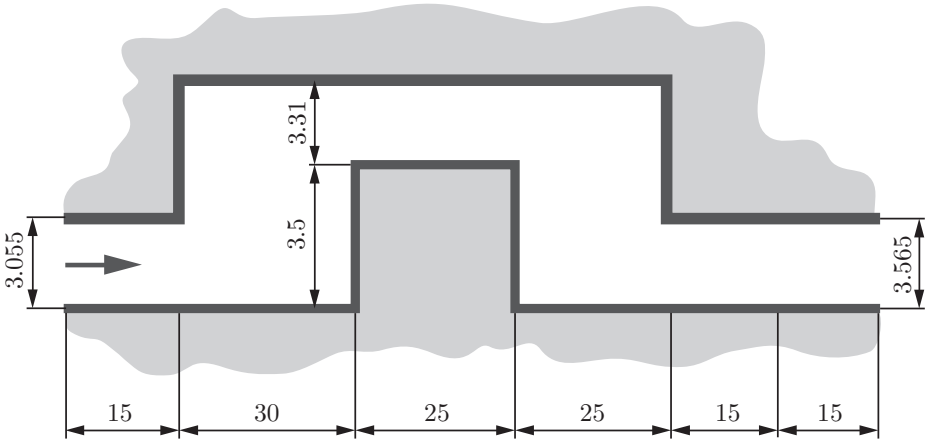


Figure 3.29: Sketch of the DLC track (in meters).

Lane-change maneuvers are considered, however, a subjective way of measuring vehicle stability, and thus there is no official regulation to this end. Only track characteristics are normalized. These maneuvers depend greatly on the driver commands and are not easily handled by closed-loop driver models. Besides, although designed for lateral stability assessment, longitudinal dynamics can affect the test and scatter the results. Yet, they are very commonly used in dynamic simulations of vehicles.

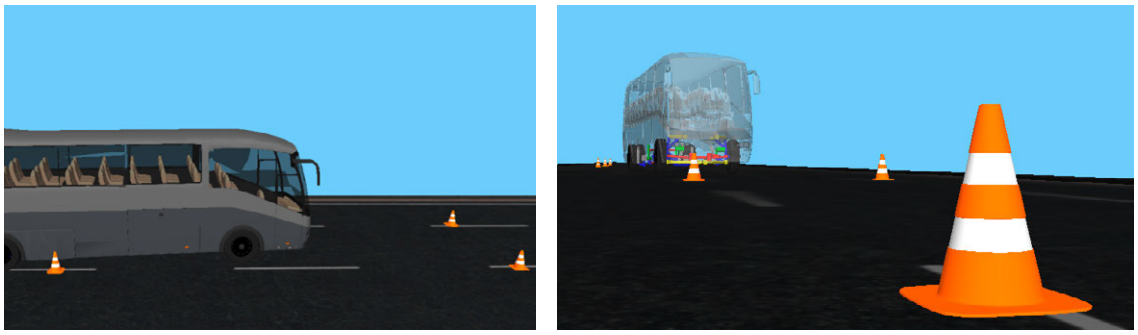


Figure 3.30: 3D-views of the DLC maneuver.

International Standard ISO 3888 describes the optimal track characteristics to perform two different severe lane-change tests: the *double lane-change* (DLC) maneuver and the *obstacle avoidance* maneuver. They are quite similar, only with different track geometries and test conditions. Strictly speaking, ISO 3888 maneuvers are only applicable to passenger cars and light commercial vehicles. However, in this Thesis they are used as a subjective way of measuring the lateral stability of the coach model. Specifically, the DLC maneuver has been cho-

sen because it is more spacious than the obstacle avoidance maneuver, and thus more suited for a large vehicle like the coach.

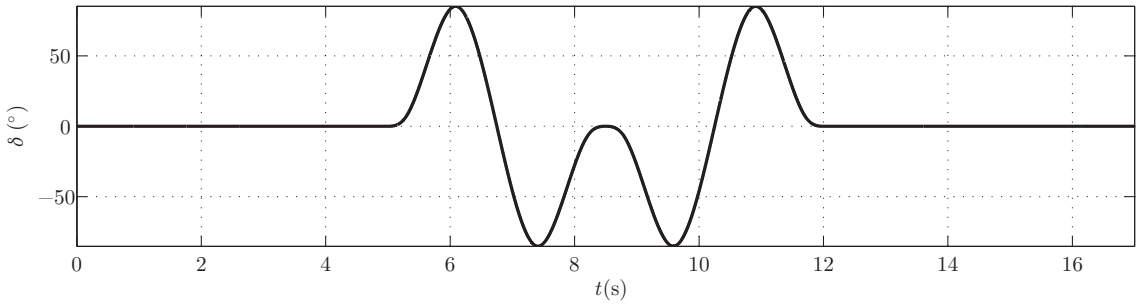


Figure 3.31: Steering coordinate function in the DLC maneuver.

A DLC simulation has been run with the coach model. Figure 3.29 shows the geometry of the double lane-change maneuver corresponding to the coach width (2.55 m). The test speed is 50 km/h, which is maintained thanks to the longitudinal controller (see Section 3.1.5). The steering function has been modeled as a spline curve with continuous first and second derivatives (that is, steering velocity and acceleration), so that the response is as smooth as possible. Figure 3.31 shows the unit steering function over time. The steering amplitude is $\delta = 0.023$ rad, starting with a left turn.

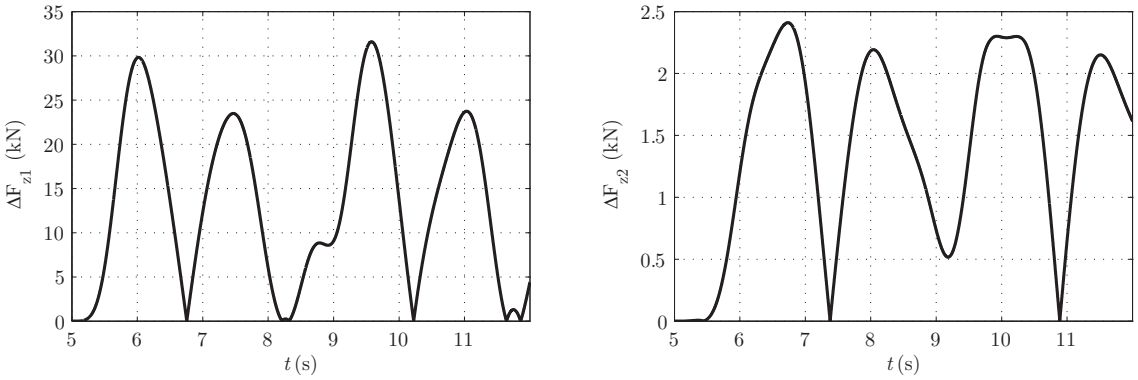


Figure 3.32: Front and rear load transfers in the DLC maneuver.

The load transfer in the DLC maneuver is shown in Figure 3.32, and the lateral acceleration and roll angle are shown in Figure 3.33. From these figures, it is clear that the lateral acceleration is far from the safety limits, and both load transfers lie within normal limits. Note that only the central part of the maneuvers is plotted in these figures, since the purpose of the first and last seconds is to facilitate the fading out of the transient effects.

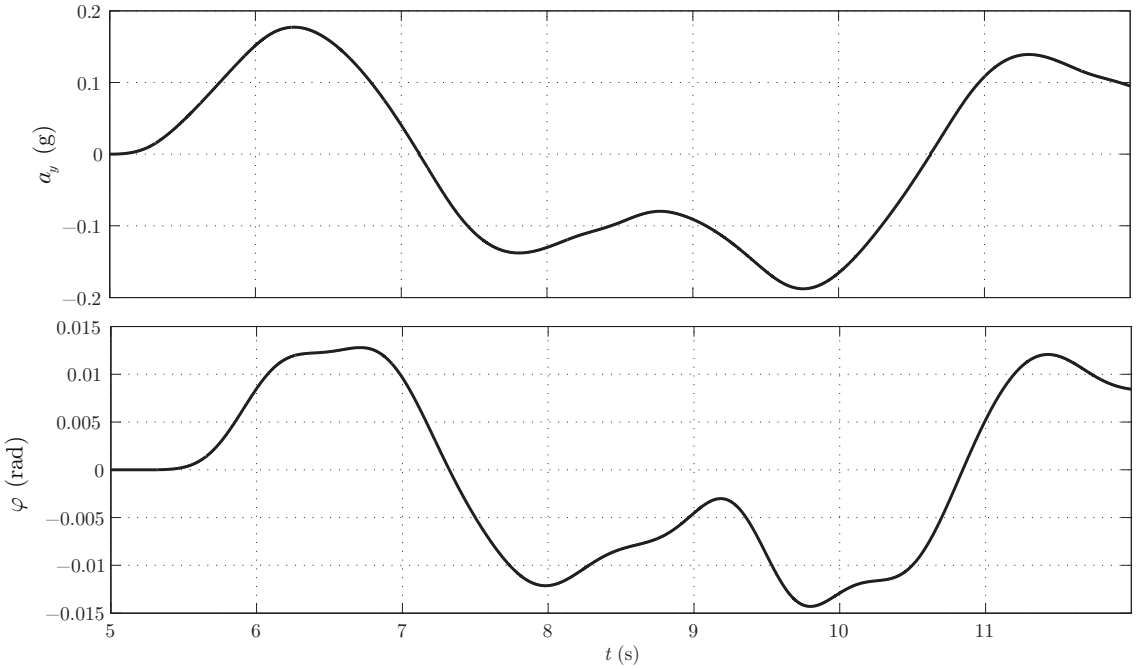


Figure 3.33: Lateral acceleration and roll angle during the DLC maneuver.

3.2.3 Step-steer test

Another typical scenario where the lateral stability can be compromised is an abrupt turn, or *step-steer* (SS) maneuver. In such a turn, the driver starts turning until a certain steering value is reached, and from that point on the steering wheel angle is kept constant. Therefore the vehicle does not return to the original lane, but instead is ridden through a transient maneuver into a steady-state turning configuration. From the amount of time taken to reach the steady state and the amplitude of the responses, interesting handling conclusions can be drawn. Step-steer maneuvers for the analysis of lateral transient response of vehicles are defined in the ISO 7401 standard. Apart from the physical characteristics of the maneuver, meaningful metrics and gradients are provided for the assessment of the response quality.

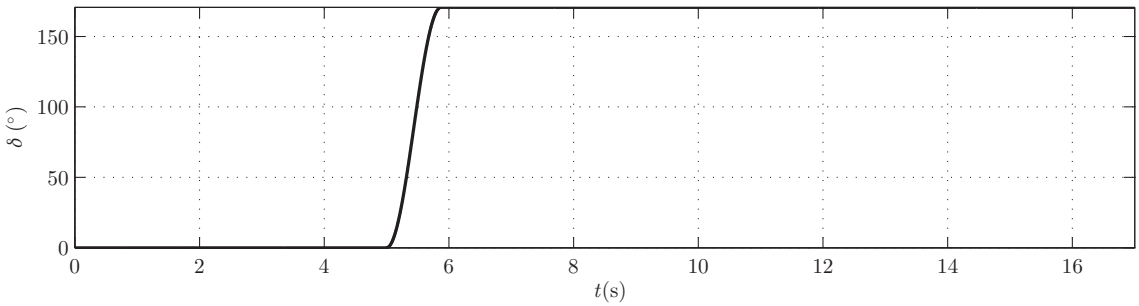


Figure 3.34: Steering coordinate function in the SS maneuver.

The step-steer maneuver has been modeled by guiding the position of the steering coordinate with a 17-second step function, as shown in Figure 3.34. As done

before with the DLC maneuver, the steering system is actuated after five seconds so that initial transient responses fade out. Figure 3.34 shows the central excerpt of the unit steering coordinate function. The corners of the position function are softened for the steering velocity to be continuous. The steering amplitude and the traction torque are the same as in the DLC maneuver. Likewise, the direction of steering is such that the vehicle turns left.

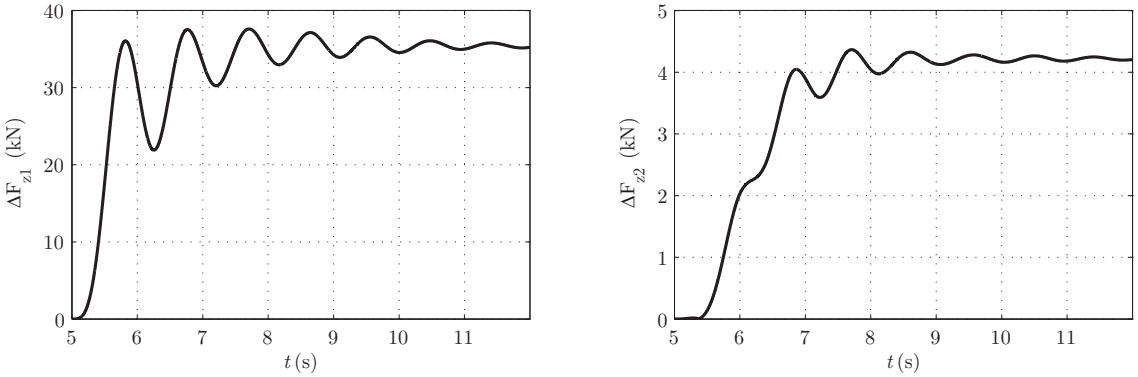


Figure 3.35: Front and rear load transfers in the SS maneuver.

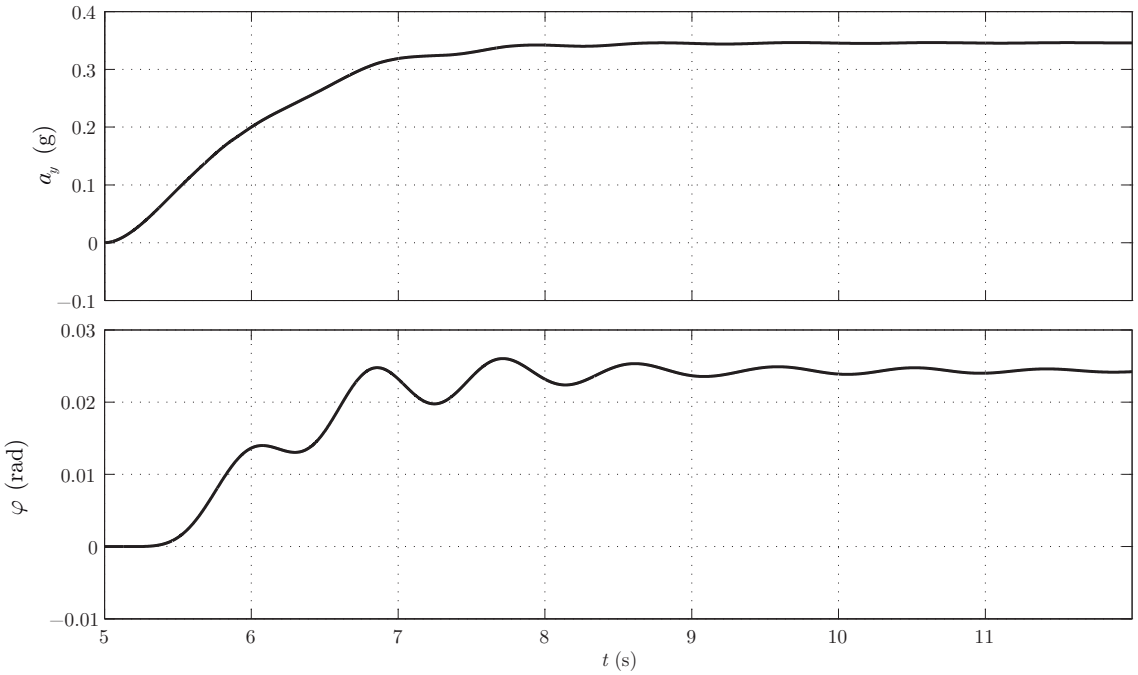


Figure 3.36: Lateral acceleration and roll angle during the SS maneuver.

Front and rear load transfers during the SS turn are shown in Figure 3.35, while the lateral acceleration and the roll angle are shown in Figure 3.36. These responses clearly show the transient nature of the test, and how, after a certain amount of time, a steady-state lateral acceleration and tire forces is reached.

3.2.4 Constant speed test

The last handling maneuver considered in this Thesis is the steady-state circular maneuver at *constant speed* (CS), as defined in ISO 4138 standard. The main purpose of this maneuver is to induce an oversteering (or understeering) behavior on the vehicle, by applying the definition of oversteering itself. Three different modes are proposed: constant radius, constant steering angle and constant speed. Here, the last one is applied. Basically, the vehicle is ridden along a circular track where the steering radius is progressively reduced, which increases the side slip differently on the front and rear axles.

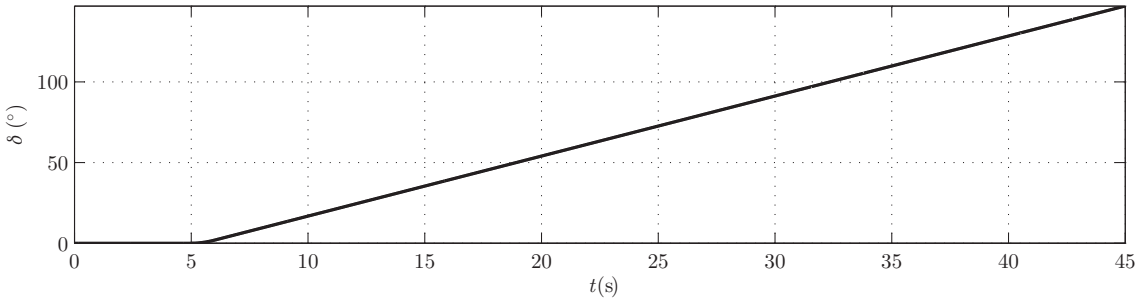


Figure 3.37: Steering coordinate function in the CS maneuver.

This maneuver was originally designed to increase the lateral acceleration in discrete steps of 0.05 g, so as to be able to reach a steady state on each step, measure the acceleration increments and extract conclusions about the oversteering behavior. However, for mathematical modeling reasons, the steering input is provided continuously as a ramp function that increases the steering angle at a similar rate (see Figure 3.37), which essentially generates the same effect on the oversteering behavior. In order to achieve significant lateral accelerations, a long simulation has to be run, as can be seen in the response figures.

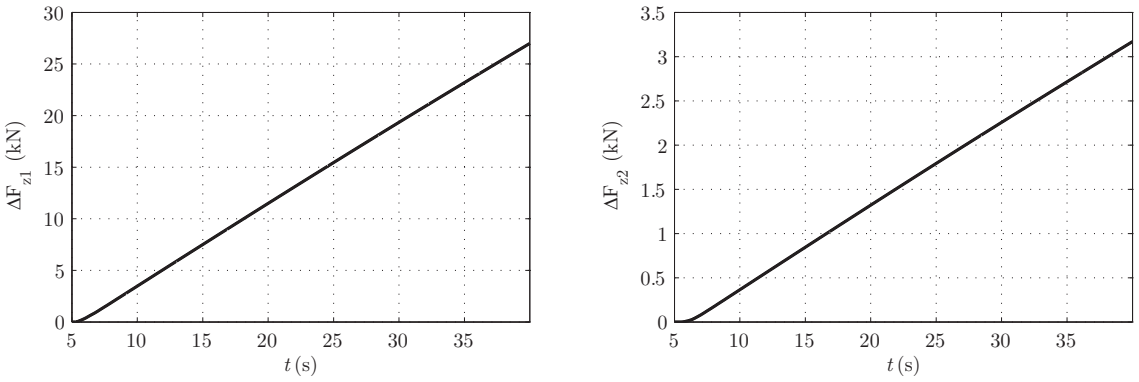


Figure 3.38: Front and rear load transfers in the CS maneuver.

As in the DLC and SS maneuvers, the front and rear load transfers are plotted in Figure 3.38. This time, instead of the roll angle, the understeering gradient $da_y/d\delta$ is plotted, together with the lateral acceleration, as Figure 3.39 shows.

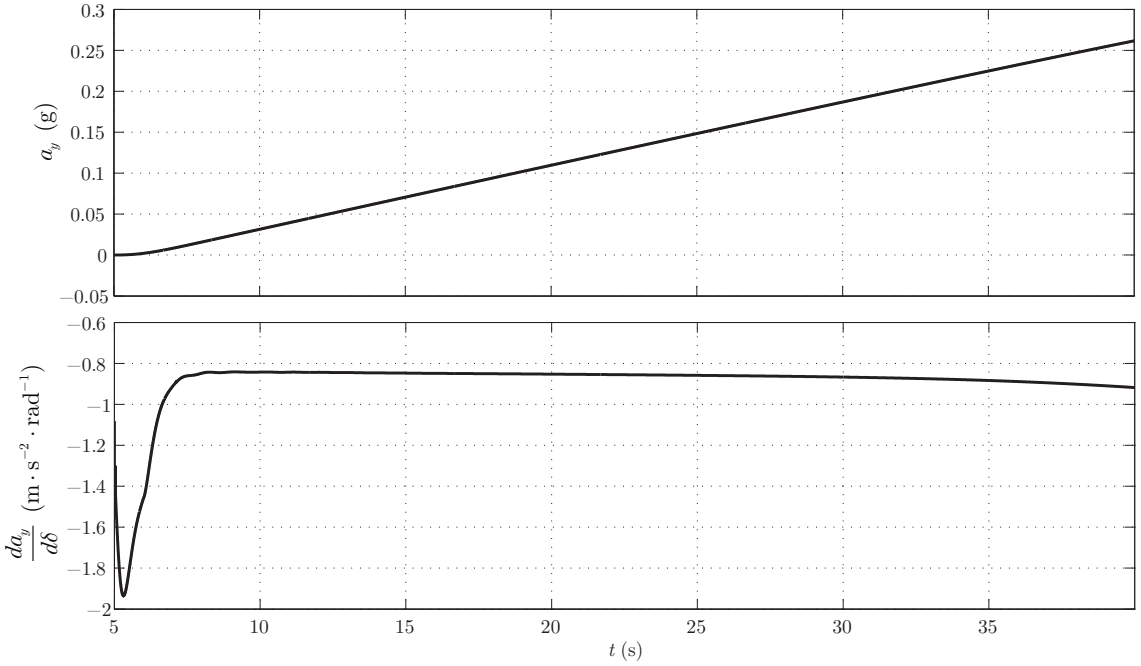


Figure 3.39: Lateral acceleration and understeering gradient in the CS maneuver.

3.3 Ride response

Comfort analyses are the natural counterpoint to handling analyses. Very often, the suspension characteristics that make a vehicle hold the road very tightly, make it frankly uncomfortable and prone to vibration discomfort. The ride response is hereby defined as the ability to filter vibration coming from irregularities in the road surface. Alongside the basic theoretical foundations of vibration discomfort, two vibration tests are presented.

3.3.1 Background

Vehicle passengers feel vibration from different sources. The evaluation of the vibration with respect to comfort is a complicated issue, and requires knowledge of different concepts such as frequencies, directions, duration, points of application of forces, etc. Even with a good understanding of the physical phenomena causing vibration, it might be difficult to define a weighted value that represents the severity of the vibration, and even more difficult to assess it. In spite of this, there have been several attempts to unify the methods for the measurement and evaluation of vibration discomfort. In this section, only discomfort coming from the feeling of vibration is evaluated. Discomfort coming from hearing movement (noise) or seeing movement is not accounted for.

In this context, discomfort vibration has to be measured at the interface of the vehicle with the passengers. Otherwise, the transmission from the measurement location and the passengers has to be estimated or measured. In the case of sit-

ting passengers, vibration comes from the contact with the floor, the seat and the seat back, which can differ greatly. However, discomfort from contact with the seat is often predominant (Griffin, 2007). Also, as high frequencies might be filtered by the seat cushion, this effect is not considered here.

Although it might be tempting to build a biomechanical model of the passenger and measure vibration in key locations of the human body, it is not useful for evaluating discomfort. The reason for this is that vibrations are felt in many different parts of the body and with different orientations, and thus it is way more practical to measure vibration only in the interfaces with the vehicle. Therefore, only whole-body vibration is considered.

As far as frequency ranges are concerned, only frequencies from 0.5 Hz to 80 Hz will be analyzed for ride comfort, according to current comfort standards BS 6841 and ISO 2631-1. Higher frequencies are usually filtered by the seat, and lower frequencies usually generate motion sickness rather than discomfort. Motion sickness will not be accounted for here because it does not directly affect comfort or handling. Each vibration frequency has a relative importance on the feeling of discomfort, and all available standards consider frequency weightings when calculating discomfort. In other words, the acceleration limit for a comfortable experience depends on the frequency of the vibration.

Accelerations at the seat-passenger interface can be translational and rotational. It turns out that the effect of roll and pitch (rotational) accelerations smaller than 0.5 Hz can be considered as an addition to translational accelerations (lateral and longitudinal, respectively). Other aspects that affect vibration discomfort are the vibration directions, the phase between vibration motions, the posture of the body, etc. Only a basic assessment of vibration discomfort is sought, and therefore these additional effects will be neglected.

There are several ways of measuring the mean value of a time-varying magnitude over time. The most basic one is the root-mean-square value of the weighted acceleration:

$$\mathrm{RMS} = \left(\frac{1}{T} \int_0^T a_w^2(t)\, dt \right)^{\frac{1}{2}} \tag{3.23}$$

where T is the duration of the measurement and a_w is the weighted acceleration. However, this equation does not account for the relationship between the magnitude and the duration of the vibration. Experience says that when the magnitude is doubled, the duration of an equivalent vibration (as far as comfort) should be divided by 16. Accordingly, a more precise meter would be the root-mean-quad value:

$$\mathrm{RMQ} = \left(\frac{1}{T} \int_0^T a_w^4(t)\, dt \right)^{\frac{1}{4}} \tag{3.24}$$

This equation gives relative importance to occasional shocks in the vibration. A more robust way of measuring discomfort would be to accumulate acceleration rather than computing the mean value. That way, the starting and ending points of measurement would be less important, and the length of vibrations would be better captured. The vibration dose value (VDV) is defined as:

$$\mathrm{VDV} = \left(\int_0^T a_w^4(t)\, dt \right)^{\frac{1}{4}} \tag{3.25}$$

VDV accounts for vibration peaks, it allows for several event VDVs to be added and is easy to use. This global value is often used as a way of quantifying vibration discomfort. However, in this Thesis, in order to make the sensitivity analysis in Chapter 5 more competitive, the RMS value has been used.

3.3.2 Four-post test

In the four-post test, the vehicle is placed on four vertical posts, which provide random displacements to each of the wheel centers independently, according to a uniformly-distributed random profile. As already explained before in the tire forces section, the implemented tire model does not account for transient behaviors when vibration is applied to the wheels. Even complete Pacejka models, like the PAC2002 Adams' model, have to be used with caution in comfort tests. For this reason, the profile is applied directly to the wheel axis (see Figure 3.40), instead of applying it to the tire surface.

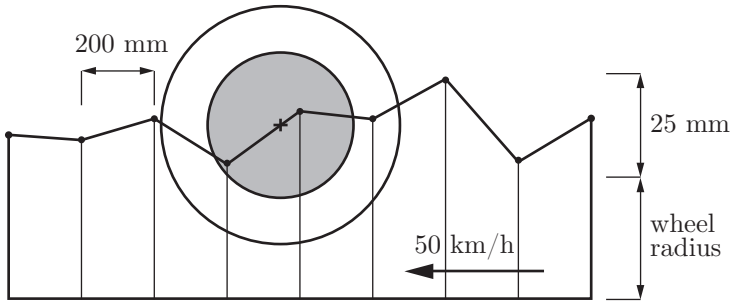


Figure 3.40: Road profile geometry for the four-post test.

The road profile is different for each wheel, and is discretized as Figure 3.40 shows. When the wheel center is between two profile points, the vertical displacement is interpolated linearly. These vertical displacements induce both vertical and roll accelerations on the bodywork, as can be seen in Figure 3.41.

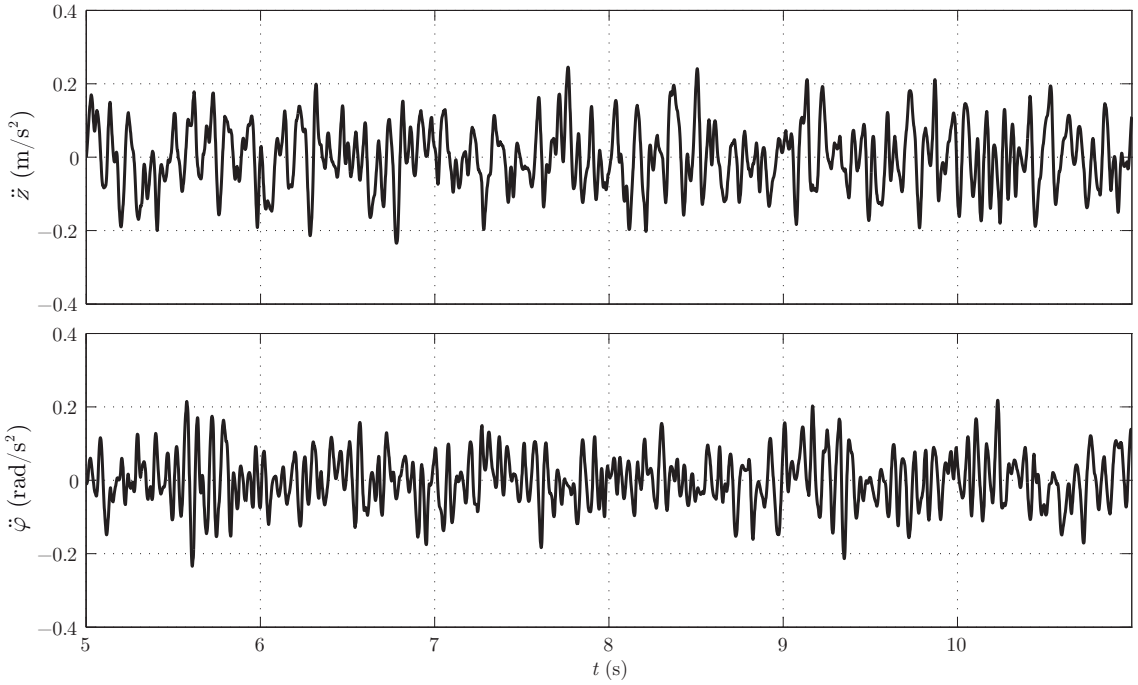


Figure 3.41: Vertical acceleration and roll acceleration in the four-post test.

An accurate analysis of the amount of vibration that reaches the passenger and the one that is filtered by the tires, the bushings, the elastic materials of the bodywork and the seat would obviously require a much more detailed model. However, multibody models have been often used as a gross estimation of the filtering ability of the suspension system. As clearly stated before, this Thesis only deals with the multi-rigid-body design of suspension systems.

3.3.3 Speed bumps test

The vertical response of the vehicle when a transient impulse is bestowed to the wheels is analyzed in this section. To that effect, the coach is simulated when riding over speed bumps. Speed bumps are a very common excitation of the suspension and also a very common source of discomfort. They are modeled as five cylindrical shapes of 2-meter radius with the axis located 1.95 m below the ground, which implies a bump height of 50 mm. They are arranged one after the other in the X-direction, with the cylinder axes parallel to the Y-direction. Bumps can be placed on the right side, on the left side, or both. In these simulations, bumps are placed on both sides and aligned laterally (see Figure 3.42) so that the main response is the pitch motion.

In order to capture the interaction with the bumps, the computation of the contact point of the wheel with the ground has to be modified. Let us assume that the toe angle of the wheel (i.e., the angle of the wheel plane with respect to the XZ-plane) is small because the vehicle is going to follow a straight path. In that case, the contact point between the wheel and the bump would belong to the

bump perimeter, which would be circular and contained in the XZ-plane. See Figure 3.43 for a schematic representation of the contact.

Figure 3.42: 3D view of the vertical impulse maneuver

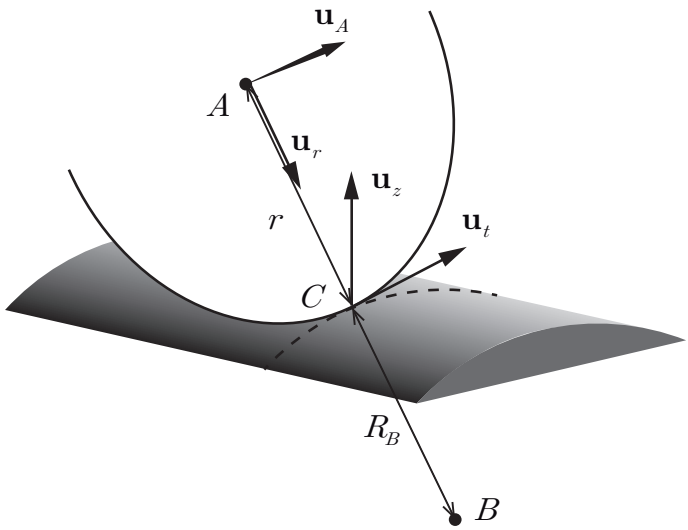


Figure 3.43: Contact point between the wheel and the bump.

From the wheel plane unit vector $\mathbf{u}_A$ and a vertical unit vector $\mathbf{u}_z$, a tangent unit vector $\mathbf{u}_t$ and a radial unit vector $\mathbf{u}_r$ can be computed. The position of the contact point would then have the following expression:

$$\mathbf{r}_C = \mathbf{r}_A + r\,\mathbf{u}_r \tag{3.26}$$

where r is the deformed radius. If we assume that the contact point $\mathbf{r}_C$ belongs to a XZ circumference of the bump, the following expression can be written:

$$(r_{Cx} - r_{Bx})^2 + (r_{Cz} - r_{Bz})^2 = R_B^2 \tag{3.27}$$

Finally, substituting Eq. (3.26) into Eq. (3.27), the value of r can be found. Due to the fact that in these comfort tests the response is going to be clearly vertical, of a relatively small frequency, and transient, a simpler approach is followed, where only the vertical acceleration of the bodywork is analyzed.

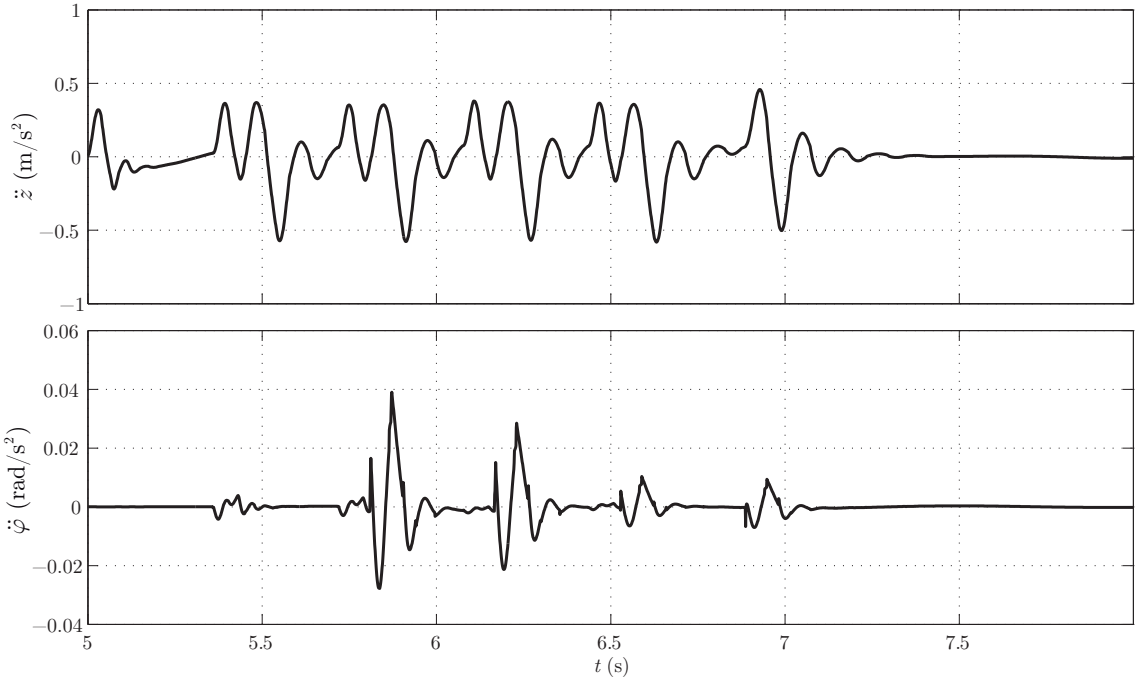


Figure 3.44: Vertical acceleration and roll acceleration in the speed-bump test.

As in the case of the random road profile, both the vertical acceleration and the roll acceleration of the chassis are analyzed, and are shown in Figure 3.44. The moments where the bumps are reached are clearly identified in the vertical acceleration chart, whereas the roll acceleration is virtually null.

Chapter 4

Automatic differentiation of dynamic variables

Automatic or *algorithmic differentiation* (AD) is a computational-mathematical technique for the computation of derivatives of computer functions (Griewank, 1989, Berz et al., 1996). Within the multibody systems literature, few attempts have been made to use AD for solving forward multibody dynamics and evaluating its computational efficiency. The most relevant implementations are found in the sensitivity analysis field, but they rarely address AD issues in depth.

In this chapter, a thorough analysis of AD tools is presented, with the main objective of assessing the advantages and disadvantages of the different AD techniques. Such a performance analysis will enable an informed application of AD tools onto two different kinds of dynamic variables: the state sensitivities in Chapter 5, and ultimately to the dynamic response optimization in Chapter 6.

As a benchmark formulation, a penalty scheme for the time integration of multibody dynamics is presented. First, open-chain generalized positions and velocities are computed recursively, while using Cartesian coordinates to define local geometry. Second, the equations of motion are implicitly integrated by using the trapezoidal rule and a Newton-Raphson iteration. Third, velocity and acceleration projections are carried out to enforce kinematic constraints. AD is tested in the computation of Newton-Raphson's tangent matrix. Specifically, the source-to-source transformation tool ADIC2 (Narayanan, Norris and Winnicka, 2010) and the operator overloading tool ADOL-C (Griewank, Juedes and Utke, 1996) are employed, in both dense and sparse modes.

Finally, the theoretical approach is supported by three examples of growing complexity: the numerical analysis of a 1-DOF spatial four-bar mechanism, three different configurations of a 15-DOF multiple four-bar linkage, and a 16-DOF coach maneuver. Numerical and AD are compared in terms of their com-

putational efficiency and accuracy. Overall, a global perspective of the efficiency of AD in the field of multibody systems is provided.

4.1 Introduction

Among the great variety of contemporary multibody formulations (Bauchau and Laulusa, 2008), penalty schemes have proven to be a robust and efficient approach for solving forward multibody dynamics using dependent coordinates (Bayo and Ledesma, 1996). Basically, they avoid the direct enforcement of kinematic constraints by introducing penalty terms proportional to the non-fulfillment of constraints. When combined with implicit integrators and projections, they allow for long integration time-steps while keeping the simulation stable and the implementation simple. One of the most interesting approaches in this direction was presented in Cuadrado et al., 2004, and Dopico, 2004, and is followed here in the preliminary stages. Natural (or fully Cartesian) coordinates are used to define local geometry and constraint equations. This approach simplifies both the modeling and the analysis stages. Positions and velocities are then computed recursively, making the most of the system topology.

For the time integration of the equations of motion, the trapezoidal rule with velocity and acceleration projections is used. This implicit scheme requires the solution of a nonlinear system of equations, which is generally solved with a Newton-Raphson algorithm. To that end, the Jacobian matrix of the open-chain forces with respect to the relative positions and velocities has to be computed. Since this step takes most of the computation time, it is worth exploring efficient and accurate ways of differentiating computer functions, while preserving the generality of the implementation.

There are several ways of computing the derivative of a mathematical function with respect to its independent variables. For example, one may apply differential calculus by hand and code the differentiated functions; this is usually called *manual differentiation* (MD). A similar but more automated technique is *symbolic differentiation* (SD), which is based on symbolic mathematical programs (such as Maple[6] and Mathematica[7]), which generate the derivative equations from the original function. These derivatives must be produced in the third-party software, exported, reimplemented, and compiled each time the equations change; and only purely analytic equations can be differentiated. Thus, this technique has considerable drawbacks from the generality point of view.

[6] Maple is a registered trademark of Waterloo Maple, Inc.

[7] Mathematica is a registered trademark of Wolfram Research, Inc.

Another way of computing derivatives is through *numerical differentiation* (ND) techniques such as finite-differences. Let f be a scalar function that depends on variable x. The derivative can be numerically approximated as:

$$f'(x_0) \equiv \frac{df}{dx}(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x} \tag{4.1}$$

which corresponds to a centered-difference formula. More accurate formulae can be obtained by evaluating the function in additional points. The advantage of these methods is that they are very simple because they only require the original function. However, Eq. (4.1) usually demands a very small value of the perturbation Δx. When Δx is very small, two very similar numbers are being subtracted in the numerator, and, because of the limited computer precision, the derivative is less accurate than the original function. These numerical errors are unavoidable. Moreover, in the case of vector functions, the computational cost increases quickly as the problem size grows.

AD allows differentiating computer functions (implemented in Fortran, C, C++, MATLAB, etc.) automatically, computing both first-order derivatives (e.g. gradients and Jacobian matrices) and higher-order derivatives (e.g. Hessian matrices). The development time of AD derivatives is shorter than using MD techniques, and these are accurate to machine precision. In past investigations with the formulation presented here, the operator overloading tool ADOL-C (Griewank, Juedes and Utke, 1996) was used successfully (Callejo and García de Jalón, 2011). However, a single AD tool was not enough for assessing the computational efficiency, since the performance of AD depends on the type of tool. Also, only academic examples were considered.

In the multibody community, very little work has thoroughly addressed AD as a way of differentiating computer functions. In 1996, Bischof used the source transformation tools ADIC and ADIFOR on a Fortran code to compute vehicle sensitivities, but general performance conclusions were not given. Three years later, Eberhard and Bischof, 1999, focused on the time integration of sensitivities using ADIFOR on a 5-DOF robot and concluded that AD was less efficient but simpler to implement than MD. Later, Dürrbaum, Klier and Hahn, 2002, proved that the symbolic tool MACSYMA generated derivatives faster than ADOL-C for two medium-size planar and spatial robots. In 2007, Ambrósio, Neto, and Leal simulated a satellite antenna as a flexible multibody system and recommended AD over ND for accuracy reasons, even though with little implementation details. Recently, Hannemann et al., 2010, applied the source transformation tool `dcc` and an operator overloading tool to dynamic models. In general, rough descriptions of AD tools and their implementation are provided; the results are not compared with other AD tools; and academic rather than industrial numerical examples are considered.

In this chapter, both the source transformation tool ADIC2 (Narayanan, Norris and Winnicka, 2010) and the operator-overloading tool ADOL-C (Griewank, Juedes and Utke, 1996) are used on three numerical examples. Namely, a 1-DOF spatial four-bar mechanism, a 15-DOF multiple four-bar linkage and a 16-DOF coach model. These examples are used as medium to large benchmarks of ND and AD tools, with special focus on computational efficiency and sparse Jacobian exploitation. As indicated before, this allows for a more experienced implementation of AD on the following chapters. Besides, to the best of the author's knowledge, the benefits of fully exploiting Jacobian sparsity in multibody formulations by using AD has not been shown before.

4.2 Semi-recursive penalty formulation

In this section, for the purpose of benchmarking AD tools, an alternative way of enforcing multibody constraint equations is presented (Cuadrado et al., 2004). The formulation is developed in three steps: (1) the loops are closed by introducing position penalty terms; (2) the trapezoidal rule of integration is introduced; and (3) velocity and acceleration projections are carried out to enforce velocity and acceleration constraints.

Let us recall the open-chain equations considered in Chapter 2, where the closed loops had been opened by cutting or removing certain joints and rods:

$$\underbrace{\mathbf{R}_d^T \mathbf{M}^\Sigma \mathbf{R}_d}_{\mathbf{M}_d^\Sigma} \ddot{\mathbf{z}} = \underbrace{\mathbf{R}_d^T \mathbf{Q}^\Sigma - \mathbf{R}_d^T \mathbf{M}^\Sigma \dot{\mathbf{R}}_d \dot{\mathbf{z}}}_{\mathbf{Q}_d^\Sigma} \tag{4.2}$$

where some terms have been grouped for the sake of clarity. These recursive equations constitute a set of n ODEs describing the motion of the open-chain system. In closed-loop systems, the constraint equations coming from the closure of the loops still need to be enforced.

Closed-loop dynamic equations can be formulated by adding the constraint equations to the open-chain dynamic equations, which have just been rewritten. The fulfillment of the position constraint equations is achieved by introducing a penalty term into Eq. (4.2). Then, velocity and acceleration constraints are imposed by carrying out velocity and acceleration projections.

First, let us add a penalty term to Eq. (4.2):

$$\mathbf{M}_d^\Sigma \ddot{\mathbf{z}} + \mathbf{\Phi}_\mathbf{z}^T \alpha \mathbf{\Phi} = \mathbf{Q}_d^\Sigma \tag{4.3}$$

where α is the penalization coefficient, $\mathbf{\Phi} \in \mathbb{R}^{m \times 1}$ is the vector of m constraint equations, and $\mathbf{\Phi}_\mathbf{z} \in \mathbb{R}^{m \times n}$ is the Jacobian matrix of the constraint equations with respect to relative positions. The penalty term has a physical meaning: $\alpha\mathbf{\Phi}$ is the value of the penalty forces (one for each constraint equation that is violated) and the columns of $\mathbf{\Phi}_\mathbf{z}^T$ are the directions of the constraint forces. Fig-

ure 2.3 shows the way a revolute closure-of-the-loop constraint can be formulated in terms of natural coordinates, as already explained in Chapter 2.

For the integration of Eq. (4.3), the implicit single-step trapezoidal rule with time-step h is used. Relative velocities and accelerations in time-step $j+1$ are written as follows.

$$\dot{\mathbf{z}}_{j+1} = \frac{2}{h}\mathbf{z}_{j+1} - \left(\frac{2}{h}\mathbf{z}_j + \dot{\mathbf{z}}_j\right) \tag{4.4}$$

$$\ddot{\mathbf{z}}_{j+1} = \frac{4}{h^2}\mathbf{z}_{j+1} \underbrace{- \left(\frac{4}{h^2}\mathbf{z}_j + \frac{4}{h}\dot{\mathbf{z}}_j + \ddot{\mathbf{z}}_j\right)}_{\hat{\ddot{\mathbf{z}}}_j} \tag{4.5}$$

By introducing Eqs. (4.4) and (4.5) in Eq. (4.3), the following nonlinear equation $\mathbf{f}(\mathbf{z}_{j+1}) = \mathbf{0}$ is obtained:

$$\mathbf{M}^{\Sigma}_{d,j+1}\mathbf{z}_{j+1} + \frac{h^2}{4}\mathbf{\Phi}^T_{\mathbf{z}_{j+1}}\alpha\mathbf{\Phi}_{j+1} - \frac{h^2}{4}\mathbf{Q}^{\Sigma}_{d,j+1} + \frac{h^2}{4}\mathbf{M}^{\Sigma}_{d,j+1}\hat{\ddot{\mathbf{z}}}_j = \mathbf{0} \tag{4.6}$$

where $\mathbf{M}^{\Sigma}_d = \mathbf{M}^{\Sigma}_d(\mathbf{z})$, $\mathbf{Q}^{\Sigma}_d = \mathbf{Q}^{\Sigma}_d(\mathbf{z},\dot{\mathbf{z}})$, $\mathbf{\Phi} = \mathbf{\Phi}(\mathbf{z})$ and $\mathbf{\Phi}_{\mathbf{z}} = \mathbf{\Phi}_{\mathbf{z}}(\mathbf{z})$.

Equation (4.6) is a nonlinear system of equations that has to be solved for unknown vector $\mathbf{z}_{j+1}$. To that end, it is customary to use the Newton-Raphson method, which has a quadratic convergence in the neighborhood of the solution. The use of this iterative method implies the evaluation of a tangent (or Jacobian) matrix and a remainder, as indicated next. Let $k+1$ be the iteration:

$$\mathbf{z}^{k+1}_{j+1} = \mathbf{z}^{k}_{j+1} + \Delta\mathbf{z}^{k}_{j+1} \tag{4.7}$$

$$\left[\frac{\partial\mathbf{f}}{\partial\mathbf{z}}\right]^k_{j+1}\Delta\mathbf{z}^{k}_{j+1} = -\left[\mathbf{f}\right]^k_{j+1} \tag{4.8}$$

The solution of Eq. (4.7) implies the evaluation of the tangent matrix in Eq. (4.8). This tangent matrix can be approximated (Cuadrado et al., 2004) with the following expression:

$$\left[\frac{\partial\mathbf{f}}{\partial\mathbf{z}}\right]^k_{j+1} \approx \left[\mathbf{M}^{\Sigma}_d + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\left(\mathbf{\Phi}^T_{\mathbf{z}}\alpha\mathbf{\Phi}_{\mathbf{z}} + \mathbf{K}\right)\right]^k_{j+1} \tag{4.9}$$

$$\mathbf{K} \equiv -\frac{\partial\mathbf{Q}^{\Sigma}_d}{\partial\mathbf{z}} \tag{4.10}$$

$$\mathbf{C} \equiv -\frac{\partial\mathbf{Q}^{\Sigma}_d}{\partial\dot{\mathbf{z}}} \tag{4.11}$$

where matrices $\mathbf{K} \in \mathbb{R}^{n\times n}$ and $\mathbf{C} \in \mathbb{R}^{n\times n}$ have been introduced. If the state vector $\mathbf{y} \equiv \{\mathbf{z}^T, \dot{\mathbf{z}}^T\}^T \in \mathbb{R}^{2n\times 1}$ is defined, both matrices can be grouped as:

$$\mathbf{J} \equiv -\frac{\partial \mathbf{Q}_d^{\Sigma}}{\partial \mathbf{y}} \tag{4.12}$$

The computation of the Jacobian matrix $\mathbf{J} \in \mathbb{R}^{n\times 2n}$ is the key step of this algorithm. Some authors (e.g. Cuadrado et al., 2004) formulate this matrix analytically, meaning that the derivatives have to be computed by hand (perhaps only in an approximated way) for the most typical force types (springs, dampers, etc.). This is obviously not the most general-purpose approach and may lead to error-prone expressions. Regarding ND, it is usually inefficient and its error is difficult to control. On the other hand, AD in its various forms can be used as well to calculate these derivatives with minimal effort from the user and reasonable efficiency. The following sections investigate the different ways of computing this Jacobian matrix.

The previous equations impose the dynamics and the fulfillment of the position constraint equations, but the velocity and acceleration constraints have not been enforced yet. During the time integration process, Eqs. (4.4) and (4.5) yield a set of velocities $\dot{\mathbf{z}}^*$ and accelerations $\ddot{\mathbf{z}}^*$ that do not satisfy velocity and acceleration constraints. The reason is that both vectors have been obtained numerically from the integrator and not by differentiating the positions. This problem can be solved through velocity and acceleration projections. The process is explained in Cuadrado et al., 2004, and is only briefly summarized here. Applying a projection method with penalty terms, one can obtain a set of velocities $\ddot{\mathbf{z}}$ that satisfy the constraints in an optimum way. Introducing a weight matrix $\mathbf{P}$, the projected velocities can be computed as:

$$\left(\mathbf{P} + \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{z}}^T \alpha \mathbf{\Phi}_{\mathbf{z}}\right)\dot{\mathbf{z}} = \mathbf{P}\dot{\mathbf{z}}^* - \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{z}}^T \alpha \mathbf{\Phi}_t \tag{4.13}$$

$$\mathbf{P} = \mathbf{M}_d^{\Sigma} + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\mathbf{K} \tag{4.14}$$

where the system matrix in the l.h.s. of Eq. (4.13) is the tangent matrix (4.9). This way, the matrix factorization of Eq. (4.8) can be reused, and the projection can be performed with a low computational cost. Similarly, the expression of the projected accelerations is:

$$\left(\mathbf{P} + \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{z}}^T \alpha \mathbf{\Phi}_{\mathbf{z}}\right)\ddot{\mathbf{z}} = \mathbf{P}\ddot{\mathbf{z}}^* - \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{z}}^T \alpha \dot{\mathbf{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}} - \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{z}}^T \alpha \mathbf{\Phi}_t \tag{4.15}$$

After solving the velocity and acceleration projections, all constraints (in position, velocity and acceleration) are fulfilled.

In standard Newton-Raphson problems, both the value and the factorization of the tangent matrix (4.9) can be reused over a number of iterations so that only a back substitution is needed to find $\Delta\mathbf{z}_{j+1}^k$ in Eq. (4.8). In this case, however,

the tangent matrix factorization is employed later to solve velocity (4.13) and acceleration (4.15) projections. These projections need an updated version of the tangent matrix (and its factorization) in order to achieve an accurate enforcement of constraints; hence, reusing the tangent matrix in the Newton-Raphson iteration is not compatible with velocity and acceleration projections. Since the computational burden of projections is greater, the author chose to always refactorize Newton-Raphson's tangent matrix and use the last factorization for projections. In turn, Jacobian matrices (4.10) and (4.11) were reused for three Newton-Raphson iterations, because they are the heaviest tangent matrix components. This approach has proven to be an effective cost-accuracy trade-off in real-life mechanical systems.

4.3 Automatic differentiation

Among the steps of Section 4.2 formulation, the computation of the Jacobian matrix in Eq. (4.12) is critical for performance. Here, we introduce AD methodology and discuss how AD is used to produce the derivatives algorithmically.

AD is an approach to obtain derivative computations based on source-code implementations of mathematical functions (Berz et al., 1996, Griewank, 1989). It combines rule-based differentiation of elementary operators (e.g. addition and subtraction) with derivative accumulation according to the chain rule of differential calculus. The derivatives produced using AD are accurate to machine precision with respect to the original computation (but not necessarily the original mathematical function; and in the case of iterative algorithms, convergence rates may differ, Griewank et al., 1993) and can be used in many contexts, including numerical optimization, nonlinear partial differential equation solvers, or the solution of inverse problems using least squares. Many tools provide AD for different languages, including Fortran, MATLAB, C, and C++ (e.g. Berz et al., 1996; Utke, 2004; Bischof, Roh and Mauer, 1997; Griewank, 1989).

AD tools typically adopt one of two implementation approaches: operator overloading (in languages that support it) or source transformation. Operator overloading-based tools are easier to implement; but since they rely on runtime evaluation of partial derivatives, the ways in which the chain rule associativity can be exploited to attain better performance of derivative codes are limited. On the other hand, source transformation approaches enable static analysis of program source code, presenting opportunities for optimization over much larger scopes than a single statement, often resulting in significantly better performance of the AD computation. Moreover, the resulting code can be tweaked and improved manually if necessary. However, source transformation-based AD has the same limitations as traditional compilers, including the complexity of implementation, parsing and analysis of general-purpose languages such as

C++, as well as reliance on necessarily conservative static analysis (e.g., alias analysis), which may lead to the generation of suboptimal derivative code.

4.3.1 Mathematical foundations

AD is based on the computational graph representation of the function under differentiation (Juedes, 1991, Griewank and Reese, 1992). Computational graphs are a type of directed acyclic graphs (DAGs). They represent the sequence of operations performed by the computer in the evaluation of a function. The result of each elementary operation is stored in a separate vertex, and the flow of variables is displayed as directed edges between vertices. Consider, as an example, the following scalar function:

$$y \equiv f(x_1, x_2) = \left(\frac{x_1}{x_2}\right)^{\sin x_2} \tag{4.16}$$

This function can be rewritten so that the result of each arithmetic operation or call to a library function is stored in a new intermediate variable x_j, with index j greater than any of the variables it depends on:

$$x_3 = \frac{x_1}{x_2} \tag{4.17}$$

$$x_4 = \sin x_2 \tag{4.18}$$

$$y = x_3^{x_4} \tag{4.19}$$

In this example, x_1 and x_2 are the independent variables, x_3 and x_4 are intermediate variables and $y \equiv x_5$ is the dependent variable. The criterion adopted to name the variables is:

- $x_1, x_2, ..., x_n$ are the independent variables.
- $x_{n+1}, x_{n+2}, ..., x_N$ are the intermediate variables used to compute dependent variables.
- $y_1 \equiv x_{N+1}$, $y_2 \equiv x_{N+2}$, ..., $y_m \equiv x_{N+m}$ are the dependent variables returned by the function.
- $x_{j,\, j>n}$ only depends on variables with index smaller than j.

In the example, $n = 2$, $N = 2$ and $m = 1$. Figure 4.1 shows the computational graph of the previous sample function. Bottom vertices are associated with independent variables, the top vertex with the dependent variable, and intermediate vertices with intermediate variables. The edges linking the vertices represent the direction of the data flow, while the value on each edge is the partial derivative of the target vertex with respect to the source vertex.

In general, a computer vector function $\mathbf{f}$ can contain the typical elements of programming languages: mathematical expressions, bifurcations, loops, library

function calls, user functions, recursive functions, etc. No matter how the function has been coded, it always starts from a set of independent variables, $\mathbf{x}$, and ends with the computation of the dependent variables, $\mathbf{y} \equiv \mathbf{f}(\mathbf{x})$, going through several intermediate operations. Function $\mathbf{f}$ can thus be decomposed into a discrete sequence of arithmetic operations (addition, subtraction, product, division) and calls to library functions (sine, exponential, etc.). AD will use this decomposition to propagate the derivatives through the code.

In the example shown in Eq. (4.16) and Figure 4.1, the derivative of the dependent variable y with respect to x_1 can be computed by applying the chain rule of differentiation systematically from the bottom vertices to the top vertex, or vice versa. Accordingly, there are two modes of AD: *forward* and *reverse*.

Forward mode In the forward mode, the chain rule is evaluated from the independent variables to the dependent variables, and the computational load is proportional to the number of independent variables. In this mode, the derivative of a vertex (i.e. of the associated variable) is the addition of the contributions of the edges that arrive to that vertex. Each edge contributes with the total derivative of the source vertex multiplied by the partial derivative associated with the edge. Thus, the derivatives can be propagated together with the evaluation of the intermediate variables, as shown next:

$$x_1' \equiv \frac{dx_1}{dx_1} = 1 \tag{4.20}$$

$$x_2' \equiv \frac{dx_2}{dx_1} = 0 \tag{4.21}$$

$$x_3' \equiv \frac{dx_3}{dx_1} = \frac{\partial x_3}{\partial x_1}\frac{dx_1}{dx_1} + \frac{\partial x_3}{\partial x_2}\frac{dx_2}{dx_1} = \left(x_2^{-1}\right)x_1' + \left(-x_1 x_2^{-2}\right)x_2' \tag{4.22}$$

$$x_4' \equiv \frac{dx_4}{dx_1} = \frac{\partial x_4}{\partial x_2}\frac{dx_2}{dx_1} = \left(\cos x_2\right)x_2' \tag{4.23}$$

$$y' \equiv \frac{dy}{dx_1} = \frac{\partial y}{\partial x_3}\frac{dx_3}{dx_1} + \frac{\partial y}{\partial x_4}\frac{dx_4}{dx_1} = \left(x_4 x_3^{\,x_4-1}\right)x_3' + \left(x_3^{\,x_4}\ln x_3\right)x_4' \tag{4.24}$$

where operator $d(.)/d(.)$ denotes total derivatives, and operator $\partial(.)/\partial(.)$ denotes partial derivatives. Note that the derivatives of the inputs are one with respect to themselves, and null otherwise. The calculations are very similar to the ones that the user would carry out by applying the chain rule by hand.

Reverse mode In the reverse mode, the chain rule is evaluated from the outputs to the inputs, and the computational load is proportional to the number of outputs. Therefore, it is especially interesting for scalar functions, where there is only one dependent variable.

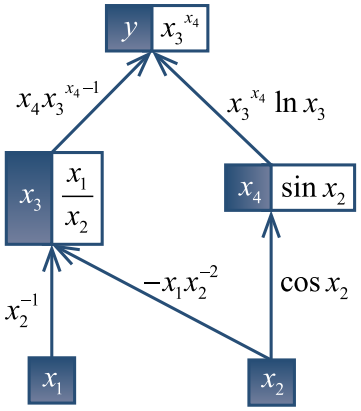


Figure 4.1. Computational graph of $y \equiv f(x_1, x_2) = (x_1/x_2)^{\sin x_2}$

In order to evaluate the computational graph backwards, two sweeps of the computational graph are required: an initial forward sweep where all the partial derivatives and the intermediate values are stored for later use, and the actual reverse sweep where the total derivatives are computed.

Upon conclusion of the reverse mode, the partial derivatives of the dependent variables with respect to the intermediate variables are available. This may be of great interest when there are several independent variables and the partial derivatives need to be computed, because the method has to be executed only once. In contrast, this mode has bigger memory requirements, because it stores all the partial results of the computational graph. The forward mode can reuse memory locations, and therefore the memory requirements are smaller.

$$\overline{y} \equiv \frac{dy}{dy} = 1 \tag{4.25}$$

$$\overline{x}_4 \equiv \frac{dy}{dx_4} = \frac{\partial y}{\partial x_4}\frac{dy}{dy} = \left(x_3^{x_4} \ln x_3\right)\overline{y} \tag{4.26}$$

$$\overline{x}_3 \equiv \frac{dy}{dx_3} = \frac{\partial y}{\partial x_3}\frac{dy}{dy} = \left(x_4 x_3^{x_4-1}\right)\overline{y} \tag{4.27}$$

$$\overline{x}_2 \equiv \frac{dy}{dx_2} = \frac{\partial x_3}{\partial x_2}\frac{dy}{dx_3} + \frac{\partial x_4}{\partial x_2}\frac{dy}{dx_4} = \left(-x_1 x_2^{-2}\right)\overline{x}_3 + \left(\cos x_2\right)\overline{x}_4 \tag{4.28}$$

$$\overline{x}_1 \equiv \frac{dy}{dx_1} = \frac{\partial x_3}{\partial x_1}\frac{dy}{dx_3} = \left(x_2^{-1}\right)\overline{x}_3 \tag{4.29}$$

In this case, the procedure is not as obvious as in the forward mode, but is equally effective. The intermediate variables are required to evaluate the derivatives, which means a forward function sweep has to be carried out first.

4.3.2 Types of tools

From the point of view of computer science, there are two main ways of implementing AD: using operator overloading tools and using source transformation tools. Both have the same theoretical foundation.

Operator overloading tools Operator overloading techniques transform each variable into a structure whose member variables are the value of the variable and the value of the derivatives with respect to the independent variables. Then, arithmetic operations and library functions are overloaded so that they handle those structures and compute both the result of the operation and the corresponding derivatives, following the rules of differentiation. The original program remains almost unchanged, because all substitutions of variables, operators and functions are carried out at compile time. This way, the implementation of AD is very simple, versatile and valid for nearly all functions. However, the execution can sometimes be slow because of the indirect costs of overloading operators and functions, namely the cost of indirect addressing and the runtime overhead. Moreover, operator overloading compilers are not as optimized as source transformation compilers.

Next, a simple example of AD through operator overloading is shown. Let us consider a product operation. To enable the calculation of derivatives, the operator has to be overloaded so that, in addition to the calculation of the product itself, the derivatives are also computed. Figure 4.2 shows a code excerpt written in MATLAB showing how to code this technique.

```
% Operator overloading
function c = mtimes(a, b)
  c.x  = a.x  * b.x;
  c.dx = a.dx * b.x + a.x * b.dx;
end
```

Figure 4.2. Overloading of product operator in MATLAB.

Variables `a`, `b` and `c` are computer structures that contain two fields: `x` and `dx`. Field `x` corresponds to the variable value, while field `dx` corresponds to the derivative or gradient (an array of length equal to the number of independent variables). The operator computes the value of `c.dx` according to the rules of differentiation of the product. The same should be applied to every other operator involving active variables. This way, when the original function is evaluated, the derivatives `dx` are propagated through the code in the form of structure field arrays, in a completely transparent way for the user.

Source transformation tools On the other hand, source transformation methods augment the code with a new piece of code containing the new variables for the derivatives and the necessary sentences to compute them. This new code usually replaces the original one, and has to be compiled and built.

If a product of two active variables was to be differentiated by using a source-transformation approach, the product operation should be replaced by the code shown in Figure 4.3, where both the function variables and the derivative variables are affected by the calculation. When applied systematically throughout the function, source transformation compilers create and inline the necessary variables to propagate the derivatives explicitly.

```
...
% Generated derivative code
c_x  = a_x  * b_x;
c_dx = a_dx * b_x + a_x * b_dx;
...
```

Figure 4.3. Source transformation of product in MATLAB.

A priori, the main advantage of source-to-source approaches over operator overloading techniques is that more work can be done at compile time, and thus the computation times can be shorter. Source transformation compilers analyze the computational graph of the original code, reorganize it and remove unnecessary operations. However, because of practical implementation limitations, this is not necessarily the case.

Two of the main disadvantages of this type of AD tools are: (1) the generated code is often cumbersome and difficult to read and debug; (2) the fact that the available source transformation compilers have not been entirely tested yet for arbitrarily complex functions, which in practice leads to a slow development. In the following subsections, additional details are given on the implementation of the derivative code, as well as on practical differences between operator overloading and source transformation tools.

Brief survey of tools There are a number of free tools for the AD of computer codes, mostly developed by universities. A simple survey has been carried out by using Google search engine, so as to get a sense of their spreading and popularity. To that end, the keywords "automatic differentiation" and the name of the tool are introduced in the search field:

```
>> "automatic differentiation" tool
```

The number of results of each search has been noted. A summary of the collected results on 2011 and 2013 can be found in Table 4.1. ADIFOR appears to be the source transformation tool with the largest presence in the bibliography, as it was the first general-purpose AD tool for Fortran codes. However, the more recent ADIC, which constitutes its C counterpart, seems to be catching up and has gained considerable presence in the last years. The rest of the source transformation tools seem to have a smaller impact. In this Thesis, ADIC2 has been used as an example of source transformation tool because all programs have

been written in C/C++. Note that the implementation of ADIC2 required close collaboration with their developers.

As far as operator overloading tools are concerned, ADOL-C for C codes is the most widespread library, with a wide presence in many engineering applications. The good documentation available enables an out-of-the-box implementation of AD techniques, which cannot be said from other AD tools. For these reasons, ADOL-C has been used throughout this Thesis for the operator-overloading computation of derivatives and sensitivities.

Source transformation				**Operator overloading**			
Tool	2011	2013	Var.	Tool	2011	2013	Var.
ADIFOR	12,400	12,600	2%	ADOL-C	9,370	9,420	1%
ADIC	11,400	22,700	99%	Fadbad	1,160	4,260	267%
Tapenade	4,270	4,130	-3%	CPPAD	1,100	3,660	233%
OpenAD	2,100	2,140	2%	SACADO	643	5,280	721%
dcc	n.a.	8,600	n.a.	Rapsodia	n.a.	3,110	n.a.

Table 4.1: Google search results retrieved on June 2011 and June 2013.

4.3.3 Strengths and weaknesses

AD performance is closely related to the way it is implemented (namely the type of tool and the differentiation mode used). This is one of the main reasons why the bibliography sometimes does not reach an agreement regarding the convenience of using AD tools in the context of multibody systems. The conclusions on the usage of a specific AD tool when solving a specific physical problem might not be applicable to other problems or tools. Frequently in the literature, authors omit to say which type of tool and mode they have used, which is basic information for a good understanding of AD results.

Accuracy, generality and development time Due to the fact that AD is based on simple arithmetic operations and on the direct application of the chain rule of differentiation, its basic accuracy is equal to the machine precision or at least equal to the precision of the considered algorithm. AD tools do not introduce additional errors by themselves. In this regard, AD is similar to SD and MD, and superior to ND.

The most general-purpose differentiation technique is ND, because, regardless of the function's complexity, it only needs two function evaluations to compute the derivative. On the other hand, the least general-purpose technique would be MD, because of the great changes it implies when little changes are introduced in the function. In between, SD and AD have similar versatility. However, AD

is more flexible from a computational point of view, as it can deal with any type of code sentence such as recursive functions, conditional assignments, loops, etc.

Regarding the development time of the differentiated function, it is clearly related with the generality of the differentiation tool. So, again, the quickest technique to implement is ND, and the slowest one is MD. AD and SD take similar development times.

Efficiency The assessment of the computational efficiency of AD is still an active research topic, especially because new ideas and AD techniques continue to come out. For augmented information on this topic, see Griewank, 1993, and Griewank and Walther, 2008. In this section, particular attention is paid to applications within the field of multibody dynamics and mechanical engineering design, although some general guidelines are given as well.

The computational cost of the forward mode is, a priori, between 2 and 2.5 times as costly as evaluating just the function:

$$\text{cost}(\mathbf{f}') \in [2, 2.5]\,\text{cost}(\mathbf{f}) \tag{4.30}$$

This cost is very close to that of computing derivatives by differencing (ND), where at least two functions evaluations have to be performed (see Eq. (4.1)). However, if ND directions are chosen so that they all share the same initial point, the forward AD propagation of derivatives is 50% more expensive than ND. This would be the worst case scenario. On the other hand, with the reverse mode, the cost of evaluating the derivatives is between 3 and 5 times the cost of evaluating just the function:

$$\text{cost}(\mathbf{f}') \in [3, 5]\,\text{cost}(\mathbf{f}) \tag{4.31}$$

The forward mode is usually more appropriate for routines in which the number of dependent variables is much larger than the number of independent variables. In turn, the reverse mode is beneficial when the number of independents is greater than the number of dependents. Although not as important these days, it should be noted that the memory requirements of the forward mode are approximately linear with the number of independent variables, while in the reverse mode they are much larger.

It must be said though, that the specific problem characteristics and the implementation can greatly affect the performance. Also, very often in mechanical problems, Jacobians and Hessians can be obtained at a smaller cost, because they are sparse or otherwise structured. In these cases, more sophisticated techniques involving sparsity exploitation and compression algorithms can improve the performance of AD in an automated fashion. The following sections include details on how to exploit such sparsity in real-life multibody problems.

Let us focus now on some bibliographical experiences. In the literature, some authors agree on the fact that AD outperforms ND, especially if the ND method

requires more than two evaluations of the function. Some examples from Barthelemy and Hall, 1992, Bischof, 1996, Anderson and Hsu, 2002, and Bischof et al., 1996 report speedups over ND by using AD. However, the opposite behavior is also found in the literature. Bischof et al., 1996, mention that sometimes the use of AD implies a loss in performance with respect to ND. Eberhard, Schiehlen and Sierts, 2007, select ND over AD. In a lot of the previous work, the results were obtained by using ADIFOR (see Bischof et al., 1995), which is a source transformation tool for Fortran codes. Whether they have used forward mode of differentiation or reverse mode is not always clear.

As far as the performance of AD w.r.t. MD, authors generally agree that AD is slower than MD (for example Barthelemy and Hall, 1992, Eberhard and Bischof, 1999, Morandini, 2006), especially if the manual derivatives are optimized. However, in rare cases AD can actually be faster than MD (see Barthelemy and Hall, 1992). As it has been previously mentioned, the main drawback of MD is that it might be very difficult to apply to complex functions, or at least very time-consuming and error-prone.

Regarding SD, several authors state that AD requires less memory than SD (Stadler and Eberhard, 2001, Anderson and Hsu, 2002). However, at least in the field of multibody system dynamics, no efficiency comparisons have been carried out, and the memory expense may be strongly connected to the specific AD tool implemented. In either case, especially within multibody dynamics, the efficiency of AD does not always follow general rules, according to Pi, Zhang and Chen, 2012, and the experience of this Thesis.

Summary of differentiation techniques In order to have an overall idea of the different ways of differentiation and their characteristics, a list has been put together in Table 4.2. The four types of differentiation (manual, MD, numerical, ND, symbolic, SD, and automatic, AD) are sorted according to four key characteristics (accuracy, generality, development time and efficiency) in a decreasing level of quality. Techniques listed in the same line have approximately the same level.

Accuracy	Generality	Development time	Efficiency	
MD, AD, SD	ND	ND	MD, SD	Better
ND	AD	AD, SD	AD	
	SD	MD	ND	
	MD			Worse

Table 4.2. General characteristics of differentiation techniques.

AD is well positioned in most of the concepts. Obviously the particular application will determine which characteristics are important and which are not. If the development time is highly constrained and the accuracy and performance of the derivatives are not a priority, the ND approach is always the simplest one.

In contrast, if accuracy or efficiency are crucial, the user should choose one of the other three techniques. Also, if the particular code is very complicated or large, MD and SD differentiation may be infeasible. As far as efficiency goes, some general ideas about AD vs. ND have already been outlined; in the next sections, a set of real examples is going to be analyzed.

Regarding the different types of AD tools (source transformation, ST, and operator overloading, OO) and modes (forward and reverse modes), they can also be rated according to the desirable characteristics: low memory usage, easy computation of partial derivatives, generality, short development time and high efficiency. Table 4.3 shows a rough ranking of tools and modes in decreasing level of quality. The number of independent variables is supposed to be twice the number of dependent variables, which is the case in the multibody formulation considered in this chapter (see Eq. (4.12)). Note that this assessment is partly based on the personal experience of the author.

Memory usage	Partial derivatives	Generality	Development time	Efficiency	
Forward Reverse	Reverse Forward	OO ST	OO ST	ST, OO	Better ⇩ Worse

Table 4.3. Characteristics of tools and modes of AD.

The "generality" and "development" time columns may be the most subjective ones, because they depend on the way the AD tools have been programmed to a great extent. The memory load might not be an issue because of the power of current computers and low cost of random-access memory. Finally, big differences in efficiency can be obtained by varying the tool and mode of AD.

4.4 Source-to-source implementation

ADIC2 is a source-to-source transformation AD tool for C and C++ with support for the forward and reverse modes (Narayanan, Norris and Winnicka, 2010). It is part of the OpenAD framework, illustrated in Figure 4.4. The input code is passed to the ROSE compiler framework (Schordan and Quinlan, 2003) which is parsed by EDG C/C++ parsers. Once converted into a ROSE abstract syntax tree (AST), the following processes occur to generate derivative code:

1. Canonicalization. Several code constructs are simplified in order to make the later transformations feasible. For example, all function calls determined to affect the output are converted into subroutine calls.
2. Program analysis. The OpenAnalysis framework (Strout, Mellor-Crummey and Hovland, 2006) is used to analyze the canonicalized code. It generates a

call graph, a control flow graph, define-use and use-define chains, a scope hierarchy, and alias analysis results.

3. XAIF generation. The results generated by OpenAnalysis and any code statements that affect the output are converted into the XML Abstract Interface Form (XAIF), a language-independent format to represent code.
4. Derivative propagation. xaifBooster uses transformation algorithms to convert the input XAIF into derivative XAIF (AD-XAIF).
5. Conversion of AD-XAIF. The AD-XAIF is parsed and is converted into ROSE AST nodes.
6. Generation of derivative code. The ROSE AST is converted into C/C++ using ROSE's `codegen` facility. The output code can then be compiled with a runtime library provided by ADIC2 and executed to generate derivatives.

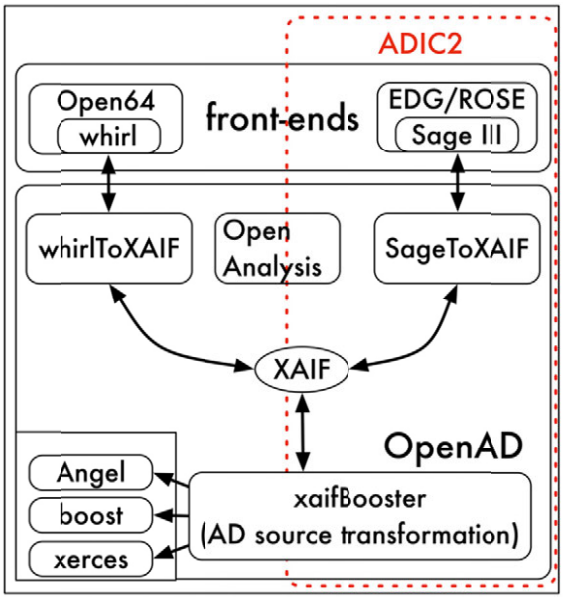


Figure 4.4: OpenAD component structure and source transformation workflow (courtesy of Krishna Narayanan).

Figures 4.5 through 4.7 show an example of the use of ADIC2. Figure 4.5 is the classic example attributed to Speelpenning, and Figure 4.6 is the corresponding output generated by ADIC2. The ADIC2 runtime library defines a structure called `DERIV_TYPE` (shown in Figure 4.7) that contains a value field and an array field that holds derivative values. The size of the array field is the number of independent variables. Operations to manipulate the array fields are defined within the library as well. At the end of the computation, the array fields of the dependent variable form the Jacobian matrix.

```
void speelpenning(double *y,double *x,int n){
  int i;
  *y = 1.0;
  for(i=0; i<n; i++) {
    *y  = (*y) * x[i] ;
  }
}
```

Figure 4.5: ADIC2 example: original function.

```
#include "ad_types.h"
#include "ad_grad_saxpy-n_dense.h"
void ad_speelpenning(DERIV_TYPE *y,DERIV_TYPE *x,int n){
  int ad_i;
  DERIV_val(*y) = 1.00000;
  ADIC_ZeroDeriv(*y);
  for (ad_i = 0, ad_i = 0; ad_i < n; ad_i = ad_i + 1) {
    DERIV_TYPE ad_prp_1;
    ADIC_Initialize(&ad_prp_1);
    DERIV_TYPE ad_prp_0;
    ADIC_Initialize(&ad_prp_0);
    double ad_lin_1;
    double ad_lin_0;
    ad_lin_0 = DERIV_val(x[ad_i]);
    ad_lin_1 = DERIV_val(*y);
    DERIV_val(*y) = DERIV_val(*y) * DERIV_val(x[ad_i]);
    ADIC_SetDeriv(*y,ad_prp_0);
    ADIC_SetDeriv(x[ad_i],ad_prp_1);
    ADIC_Sax_2(ad_lin_0,ad_prp_0,ad_lin_1,ad_prp_1,*y);
  }
}
```

Figure 4.6: ADIC2 example: differentiated function.

```
typedef struct {
 double val;
 double grad[ADIC_GRADVEC_LENGTH];
} DERIV_TYPE;
```

Figure 4.7: ADIC2 example: derivative structure.

Since Jacobians can be sparse, using an array size that effectively computes a full Jacobian can be inefficient. Furthermore, for large Jacobians, not enough memory may be available to allocate an array for each `DERIV_TYPE` variable. Therefore, ADIC2 implements a framework to exploit Jacobian sparsity (Narayanan et al., 2011). Specifically, given a function $\mathbf{Q}_d^{\Sigma}(\mathbf{y}) : \mathbb{R}^{2n\times 1} \rightarrow \mathbb{R}^{n\times 1}$ whose Jacobian matrix $\mathbf{J} \in \mathbb{R}^{n\times 2n}$ (see Eq. (4.12)) is sparse, ADIC2 employs the following framework to efficiently compute matrix $\mathbf{J}$ using these steps:

1. Determine the *sparsity pattern* of matrix $\mathbf{J}$.
2. Using a *coloring* on an appropriate graph of $\mathbf{J}$, obtain an $n \times p$ *seed matrix* $\mathbf{S}$ with the smallest p that defines a partitioning of the columns of $\mathbf{J}$ into p groups.
3. Compute the numerical values in the *compressed matrix* $\mathbf{B} \equiv \mathbf{JS}$.
4. Recover the numerical values of the entries of $\mathbf{J}$ from $\mathbf{B}$.

In step 1, the output derivative code is compiled with a runtime library called SparsLinC, which is used to detect the structure of the Jacobian. In step 2, the coloring package ColPack (Gebremedhin et al., 2011) is used. The number of colors p used to partition the Jacobian dictates the number of columns in the compressed matrix and consequently the new size of the array in the `DERIV_TYPE` structure. When computing the compressed matrix, having a smaller array can result in a performance improvement, provided that the overheads of the steps 1, 2, and 4 can be offset.

4.5 Operator-overloading implementation

Throughout this Thesis, ADOL-C library has been used for the operator overloading differentiation of computer functions. As already mentioned, it is the most popular AD tool for C/C++ codes. It was developed by Walther and Griewank, 2012. It can be downloaded and used under the Common Public License (CPL) or the GNU General Public License (GPL). ADOL-C stands for Automatic Differentiation by Overloading in C++.

ADOL-C can evaluate first and higher-order derivatives of C/C++ functions by using the operator overloading technique. It works both with forward and reverse modes of differentiation. The error of the computed derivatives is kept to machine precision. Derivatives can be computed in dense mode or in sparse mode. The cost of evaluating derivatives and higher-order tensors grows with the square of the derivative order, but is independent from the matrix size. ADOL-C implements multiple easy-to-use routines and drivers for the forward and reverse evaluation of derivatives, the solution of ODE systems, dependency analyses, etc. The latest improvements are the sparse pattern detection in Jacobian and Hessian matrices, the compression of sparse matrices, the checkpointing of time integration algorithms, the usage of fixed point iterations and the differentiation of OpenMP codes.

Types of variables Before using ADOL-C differentiation functions, a few changes have to be introduced in the code. The main concept when implementing ADOL-C is the concept of active and passive variables. Passive variables contain only the numerical value they represent, whereas active variables contain the numerical value and the derivative of the variable with respect to the

independent variables (i.e., the inputs of the function). These derivatives are then propagated through the code using operator overloading techniques. All operations that involve an active variable need to be overloaded so that they can handle the operations with the derivatives. There are three reasons why a C variable may be classified as active:

– It belongs to the set of dependent variables of the function, and the user may request its derivative with respect to any of the independent variables.
– It belongs to the set of independent variables, and the user may request the derivative of any of the dependent variables with respect to it.
– It depends mathematically on an active variable, and thus needs to be active to enable derivative propagation.

On the other hand, passive variables do not contain any derivative information, and act as constants in the computer function. They can take part in the definition of active and passive variables, but cannot be defined from active variables. Only active variables can be defined from active variables. Also, the derivative of an active variable with respect to a passive variable cannot be computed, and a passive variable cannot be differentiated.

The user must specify which C/C++ variables are active by declaring them as an `adouble` data type. This data type is defined at ADOL-C's header file `<adolc/adouble.h>`. From the moment they are defined, those `adouble` variables will propagate the value of the derivatives throughout the code. The variable value can be accessed at any time as `variable.value()`. On the other hand, passive variables are always of `int`, `float` or `double` type. All basic C/C++ functions (`fabs`, `fmin`, `fmax`, `pow`, `sqrt`, etc.) and operators (`++`, `--`, `+=`, `-=`, `<<`, `>>`, etc.) are overloaded so that they handle active arguments.

All operations and sentences involving active variables make up the computer function under differentiation. To clearly define the beginning and the end of the piece of code under differentiation (which might be only a part of the global code), tags `trace_on(0)` and `trace_off()` are introduced, respectively. All operations between those tags (which, in fact, constitute the computational tree), are registered in a data structure called the *tape*. The tape is used later on to evaluate the derivatives and even the function itself.

Among active variables, there is a set of independent variables (mathematical inputs) and a set of dependent variables (mathematical outputs) which constitute the mathematical interface of the function. ADOL-C "knows" about these sets through the use of `<<=` and `=>>` operators, respectively. The active variable is assigned the numerical value of the passive variable on the r.h.s. of the operator, and is registered as an input with respect to which the outputs can be differentiated. Similarly, operator `=>>` is used at the end of the active code to specify the outputs or dependent variables. This time, the r.h.s. of the operator

is a passive variable that is assigned the value of the active variable, as the direction of the inequality symbols indicate. Figures 4.8 and 4.9 show how ADOL-C handles active variables in the Speelpenning example.

```
#include "adolc/adolc.h"
void speelpenning(double *yp,double *xp,int n){
  adouble *x = new adouble[n];
  adouble  y = 1;
  trace_on(0);
  for(int i=0; i<n; i++) {
    x[i] <<= xp[i];
    y *= x[i];
  }
  y >>= *yp;
  delete[] x;
  trace_off();
}
```

Figure 4.8. ADOL-C example: definition of the function.

```
#include "adolc/adolc.h"
void speelpenning_driver(double* g,double *xp,int n){
  double* g = new double[n];
  gradient(0,n,xp,g);
}
```

Figure 4.9. ADOL-C example: computation of the gradient.

Following the same notation as in previous subsections, the problem under consideration requires the computation of Jacobian matrix $\mathbf{J} = \partial \mathbf{Q}_d^{\Sigma}(\mathbf{y}) \,/\, \partial \mathbf{y}$. This means function $\mathbf{Q}_d^{\Sigma}(\mathbf{y})$ has to be taped. To this end, an ADOL-C version of the aforesaid computer function has to be written, where both the outputs and the inputs are declared as active, as does any intermediate variable that depends on an active variable. Constants and other side variables remain passive.

Tape evaluation In practice, implementation problems arise when trying to declare active variables in the code. Basically, all user-defined functions have to be adapted to the new datatype `adouble`, including mathematical operations, memory allocation functions, etc. Probably the only way of keeping the active and passive versions of the code in a maintenance-friendly and general way is to duplicate the code and physically keep two separate versions. This is obviously a drawback, especially when the code is under continuous development, which is very common in a research context. Other approaches like switching from the passive pieces of code (for instance function headers) to the active ones by using `#include` statements are strongly discouraged by the C/C++ community, as they make the code difficult to read and maintain.

On the other hand, the user should try to declare as few active variables as possible, since they take up more memory and runtime operations than passive variables (note that active variables are nothing but computer structures). Howev-

er, when the code is large and complicated, it is hard (or even impossible) to track which variables depend on active variables and which variables do not. This is especially true when very numerous intermediate variables are declared in the functions and recursive functions are used, which is the case here. Unfortunately, there is a lack of general-purpose tools for the dependency analysis of active variables.

These obstacles often lead to the development of two parallel versions of the code: the active one and the passive one. In the active one, all `double` variables are transformed to the `adouble` datatype, except the ones that are clearly passive (physical constants, etc.). This strategy involves declaring many more active variables than are probably necessary, but is the only automatic approach.

That said, the preparation steps can be summarized as:

1. Introduce sentences `trace_on(tag)` and `trace_off()` at the beginning and end of the active section.
2. Declare dependent and independent variables, and all those which depend on them, as `adouble`.
3. Set the value of the independent variables from the value of passive variables by using the `<<=` operator.
4. Assign the value of the dependent variables to the corresponding passive variables by using the `=>>` operator.
5. Include header file `<adolc/adouble.h>` and compile the code.

Once the code has been prepared, the tape can be generated by running the code, and later on used to compute the derivatives, as explained next.

The tape stores the computational graph of the piece of code under differentiation, that is, a registry of all mathematical operations. It is created the first time the function is evaluated, and can be physically written to disk or kept in memory. This sequence of basic operations can then be used to evaluate the derivatives and/or the function. The capability of evaluating the function by using the tape can be useful to port the function to other computer environments. Besides, the evaluation of the tape can be faster than the evaluation of the original function, since the tape is written in a very efficient way.

Note that the only operations recorded by the tape are the ones executed when the function was run for the first time. This involves a potential danger: if the point where the tape is evaluated is not the same as the point where the tape was generated, and the computational flow of the program is different (for example, due to conditional sentences and loops), the evaluation of the tape will yield wrong results. This problem is well-known in tape-based operator overloading libraries. As an example, let us differentiate the following sample code:

```
if (a1>3){
 a2 = b*c;
}else{
 a2 = b/c;
}
```

If the tape registered only the first sentence of the conditional assignment (i.e., if `a1>3` when the function was first run), the second part of the loop is not recorded, and therefore the tape will return wrong results when `a1!>3`. Thus, the condition for the validity of the tape is that the flow through conditional assignments (including `if`, `switch`, etc.) coincides with the one registered in the tape (corresponding to the first execution of the code). The same can be applied to the computation of derivatives. Derivatives will be propagated incorrectly through the tape if the flow is not the original one.

Obviously, this behavior only affects conditional assignments involving active variables depending on independent variables, which are the ones that will change from one execution to another. The rest of variables will be registered as constants in the tape, and therefore cannot alter the execution flow. However, AD developers knew from the beginning it was of utmost importance to solve this issue, since many computer codes contain conditional assignments.

There are two workarounds to avoid this undesirable effect. The first one is called *retaping*. It basically consists of re-recording the tape every time the execution flow has changed. To this end, ADOL-C drivers return an output value to let the user know that the flow has changed. However, this strategy is too expensive, as the generation of the tape is far less efficient than the evaluation of the tape. The second approach consists of recording all possible code branches in the tape, so that the tape records all operations. To this end, ADOL-C provides the `condassign` function, which is called as follows:

```
condassign(a,b,c,d)
```

which is equivalent to

```
a = (b > 0)? c:d;
```

or

```
if (b > 0){
  a = c;
}else{
  a = d;
}
```

When function `condassign` is called, both branches of the code are registered in the tape. This solves the problem of the conditional bifurcations. However, both branches have to be evaluated in the original function, which may bring

undesirable collateral effects. In practice, conditional loops can be appropriately transformed to `condassign` calls with a little experience and not too much effort. Almost all conditional statements can be adapted to the `condassign` philosophy, although sometimes multiple calls and temporal variables are required. As an example, this code:

```
if (b==1){
  a = 10;
}else if (b==2){
  a = 20;
}else{
  a = 30;
}
```

would be transformed into `condassign` calls as:

```
a = 30;
condassign(a, fabs(b-2), a, 20);
condassign(a, fabs(b-1), a, 10);
```

Another reason why the tape might stop being valid at some point is that a discontinuity in functions `fmax`, `fmin`, `fabs` is reached. Both the flow branching and the discontinuities are reported to the user by ADOL-C when calling the differentiation routines. When this happens, the tape is no longer valid and needs to be regenerated at that point. Throughout this Thesis, retaping is avoided because of its high computational cost.

Drivers Once the active section of the code is delimited and the tape has been recorded, the final step is to use the tape to actually evaluate or differentiate the original function. ADOL-C provides many different drivers for the evaluation of the tape, including gradients, Jacobians, Hessians, miscellaneous products of derivatives and arrays, functions for the solution of optimization problems and ODEs, etc. For the present application and the following ones (see Chapter 5), the most important drivers would be:

– `function(tag,m,n,x,y)` for the evaluation of the function itself, where `tag` is the tape ID, `m` is the number of outputs, `n` is the number of inputs, `x[n]` is the input array pointer and `y[m]` is the output array pointer.

– `fov_forward(tag,m,n,p,x0,X,y0,Y)` for a general evaluation of the first-order vector function derivatives in forward mode, where `tag` is the tape ID, `m` is the number of outputs, `n` is the number of inputs, `p` is the number of directions, `x0[n]` is the input array pointer, `X[n][p]` is the seed matrix, `y0[m]` is the output array pointer and `Y[m][p]=J[m][n]*X[n][p]`.

- `fov_reverse(tag,m,n,q,U,Z)` for a general evaluation of the first-order vector function derivatives in reverse mode, where `tag` is the tape ID, `m` is the number of outputs, `n` is the number of inputs, `q` is the number of weight vectors, `U[q][m]` is the weight matrix and `Z[q][n]=U[q][m]*J[m][n]`. In order to use the intermediate variables in the reverse sweep, this function requires a prior call to `zos_forward(tag,m,n,keep,x,y)` with `keep=1`.
- `jacobian(tag,m,n,x,J)` for the evaluation of the explicit Jacobian matrix, where `tag` is the tape ID, `m` is the number of outputs, `n` is the number of inputs, `x[n]` is the input array pointer and `J[m][n]` is the Jacobian matrix pointer. Depending on the dimensions `m` and `n`, the driver decides whether to use `fov_forward` (forward mode) or `fov_reverse` (reverse mode).
- `sparse_jac (tag,m,n,repeat,x,&nnz,&rind,&cind,&values,&opt)` for the evaluation of a sparse Jacobian matrix, where `tag` is the tape ID, `m` is the number of outputs, `n` is the number of inputs, `repeat` is a Boolean variable specifying whether to use or not the previously computed sparse pattern (`rind` and `cind`), `x[n]` is the input array pointer, `nnz` is the number of nonzeroes, `rind[nnz]` is the array of nonzero row indices, `cind[nnz]` is the array of nonzero column indices, `values[nnz]` is the array of nonzero Jacobian values, and `opt` is a structure containing options.

See Walther and Griewank, 2012, for further information on the available drivers. These drivers can be used anywhere throughout the code, as long as the tape has been properly generated and is stored in memory.

4.6 Results

Five different mechanical systems are simulated in order to assess the accuracy and efficiency of AD tools in the presented formulation, in collaboration with S. H. K. Narayanan. The first one is a 1-DOF spatial four-bar mechanism. The second, third, and fourth examples are three different configurations of a 15-DOF multiple-four bar linkage. The fifth is a 16-DOF coach performing a lane-change maneuver. In all cases, gravity acts in the $-Z$-direction. All simulations have been run on two different platforms. Platform gcc-1 is a dual Intel® Xeon™ (8 processors at 2.66 GHz) with 32 GB RAM running Ubuntu 12.04. Platform gcc-2 is the one detailed in Appendix A under "OS II", that is, running Ubuntu operating system.

The simulation time is 5 s in all experiments except for the coach case, where 2 s are simulated. Three different time-steps (1, 10, and 20 ms) have been tested, in order to capture the effect of the time-step length on the simulation accuracy and efficiency. To monitor the physical accuracy of the formulation, an energy balance is carried out. Kinetic and potential energies and the work of non-

conservative forces are computed over time and summed. The variation in the total energy of the system (or *energy drift*) is provided for each of the integration time-steps, showing how longer time-steps make the implicit integrator numerically competitive but physically less accurate.

System	Indeps.	Deps.	Nonzeroes	Colors
Spatial four-bar	8	4	20	6
1×15 four-bar	90	45	1410	60
4×15 four-bar	270	135	4290	60
7×15 four-bar	450	225	7170	60
Coach	66	33	1078	62

Table 4.4: Jacobian matrices for the different models.

For each system, derivatives are obtained first by employing centered-difference ND as in Eq. (4.1). Then, ADOL-C is run in the forward, reverse, and sparse modes, and ADIC2 is run in the forward and sparse modes. For each case, the time to complete the simulation, the energy drift, and the number of Jacobians computed are noted.

4.6.1 Spatial four-bar linkage

The first model under study is the 1-DOF spatial four-bar mechanism shown in Figure 4.10. The lengths of the crank, connecting rod, follower, and ground link are, respectively, $L_c = 1.5$ m, $L_r = 4$ m, $L_f = 2$ m, and $L_g = 5$ m. Each bar k has mass $m_k = L_k$ and negligible inertia around the direction of the axis. The only initial condition is $z_4(t=0) = -0.1$ rad/s. Row 1 of Table 4.4 lists the number of independents, dependents, nonzeroes, and colors used to partition the Jacobian. The size of the Jacobian in this model is quite small and is not very sparse, as can be seen in Figure 4.13(a).

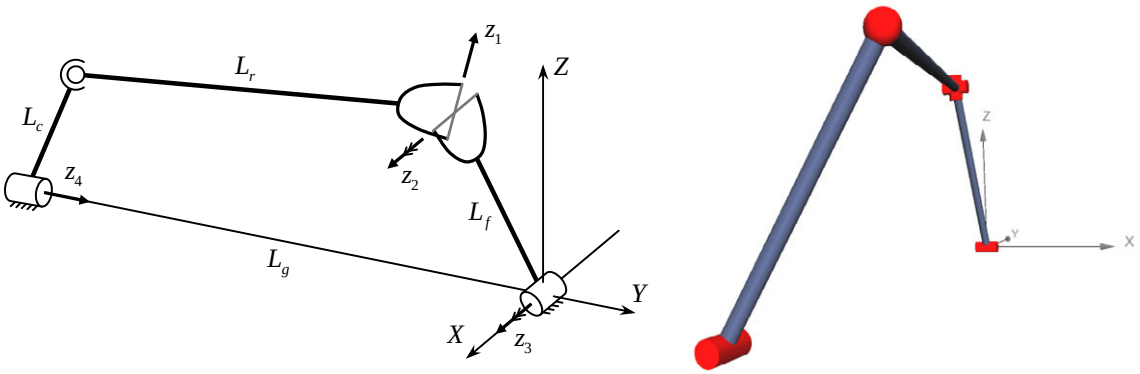


Figure 4.10: Schematic and 3D views of the spatial four-bar mechanism.

Computation times are shown in Table 4.5(a). Columns contain the elapsed times of the different differentiation methods (ND, AD with ADOL-C, and AD with ADIC2), and rows contain the different platforms used in the simulations.

In this example, elapsed times are at the limit of what can be measured with standard timing functions. Nevertheless, a trend can already be observed: AD times are about the same as ND times, and sparse modes seem to be the fastest way of running AD. However, differences in AD modes and tools are not yet clearly quantifiable.

4.6.2 Multiple four-bar mechanism

The second sample model is a multiple four-bar linkage, made up of a series of concatenated four-bar mechanisms in the Y- and Z-directions. The number of quadrilaterals in both directions is denoted as n_y and n_z, respectively. Three different $n_y \times n_z$ cases are considered: 1×15, 4×15, and 7×15. See Figure 4.11(a) for a generic case and Figure 4.11(b) for the 1×15 case. All joints in the system are parallel X-direction revolute joints, and all bodies are contained in the YZ-plane. Only the top joints are fixed; all bars are moving bodies. Bars have a uniformly distributed mass of 1 kg and a length of 1 m. The system is considered as a three-dimensional multibody system for the sake of generality. The only initial condition is $\dot{z}_1(t=0) = \pi\,/\,3$ rad/s. Rows 2–4 of Table 4.4 list the number of independents, dependents, nonzeroes, and colors used to partition the Jacobian for the 1×15, 4×15, and 7×15 cases, respectively. The Jacobians are sparse and the number of colors used to partition the Jacobian is considerably lower than the number of independent variables. This implies that, in this case, ADIC2's sparse mode requires much less stack memory than ADIC2's dense mode.

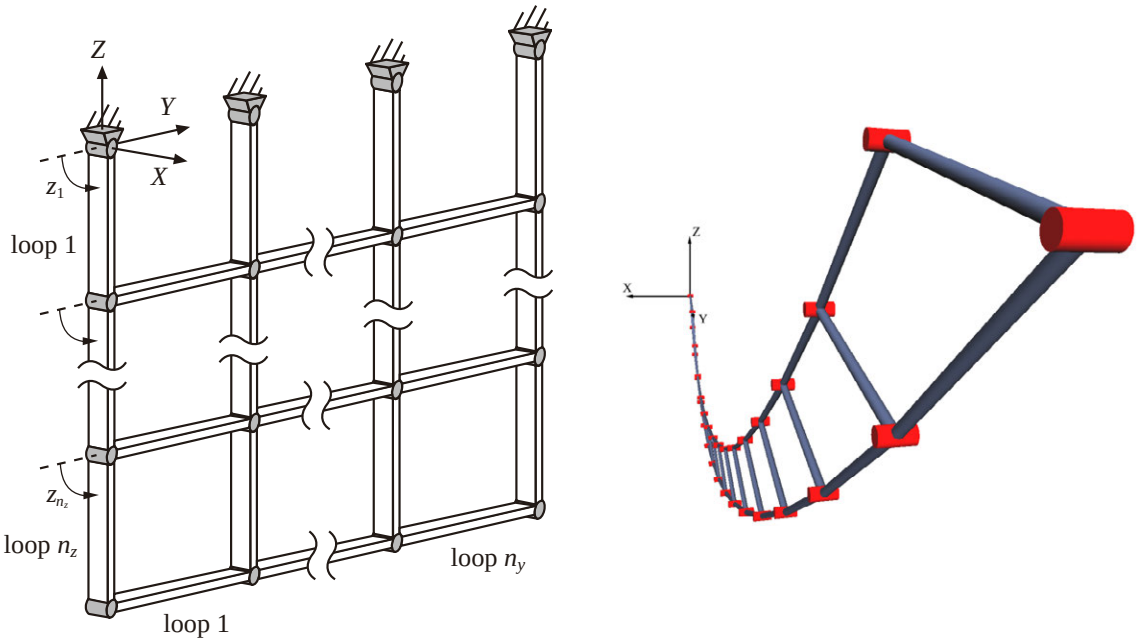


Figure 4.11: Schematic multiple four-bar linkage and 3D view of the 1×15 case.

The main objective of the multiple four-bar linkage experiments is to assess the effect of the problem size on the computational cost of Jacobian matrix (4.12). All three systems are 15-DOF systems, but they differ in the number of rigid bodies (45, 135, and 225, respectively). For a given number of DOFs, the higher the number of bodies (and thus joints), the higher the number of constraints.

Computation times are shown in Table 4.5(b). Several observations can be made for these models. First, both AD tools (ADIC2 and ADOL-C) have a very similar efficiency. Second, AD times are shorter than ND times in the 4×15 and 7×15 cases. Multiple four-bar systems with a high ratio of rigid bodies per DOF seem to favor AD performance. The reason for this is that the size of the compressed matrix is much smaller than the number of independent variables. This fact coincides with previous investigations using ADOL-C in similar contexts (Callejo and García de Jalón, 2011) and proves that ADIC2 follows the same trend. Third, due to the fact that the `grad` field of `DERIV_TYPE` is statically allocated, ADIC2-generated code exceeded the available stack space in the machine for the two infeasible cases. Fourth, ADOL-C's reverse mode is faster than ADOL-C's forward mode.

4.6.3 Coach dynamic maneuver

The coach from Chapter 3 is simulated. All parameters and model features (including topology, applied forces and suspension components) apply, except for the rear dual tires, which are hereby considered as single tires. A general view of the model is shown in Figure 4.12.

Figure 4.12: 3D views of the coach maneuver.

The guided coordinate δ corresponds to the rotation of the steering actuator, which acts on the rods through the steering mechanism. In the considered simulation, the coach performs a 2-second lane-change maneuver. To that end, coordinate δ is kinematically driven by a predefined steering function $\delta(t) = 0.013 \sin(\pi t)$ [rad], $t \in [0,2]$. The speed control is set to 50 km/h. The resulting model is a 16-DOF multibody system. Row 5 of Table 4.4 lists the number of independents, dependents, nonzeroes and colors used to partition the Jacobian, which in this model is not sparse (see also Figure 4.13(c)).

Computation times are shown in Table 4.5(c). In this case, AD times are slightly longer than ND times. Both for ADIC2 and ADOL-C, the benefits of the sparse mode are almost unnoticeable due to the dense Jacobian. Finally, ADOL-C's forward mode is faster than ADOL-C's reverse mode.

4.6.4 Discussion and conclusions

Both ADIC2 and ADOL-C were relatively easy to use and provided reliable, accurate derivatives. The multibody simulation code is the most complex use case that ADIC2 has ever been applied to, and has required several enhancements to the tool, introduced in close collaboration with the developers.

When Jacobian matrices are considered naively to be dense, ADIC2 and ADOL-C codes are always slower than ND code. This result may be due to overheads in the implementation of ADIC2's and ADOL-C's runtime libraries. Furthermore, because of the nature of Newton's method, Jacobian accuracy is not essential for rapid numerical convergence. In fact, the inherent machine-precision accuracy of AD has almost no effect on the number of iterations required by Newton's algorithm to solve the nonlinear system of Eqs. (4.6). Thus, in this formulation all methods converge in the same number of iterations.

The sparsity of the Jacobian has a vital effect on performance. Figure 4.13 shows the sparsity patterns of three models used here. The multiple four-bar linkage model is quite sparse and it has been exploited by both AD tools. While sparsity could certainly be exploited for the ND approach as well (Curtis, Powell and Reid, 1974), the multibody code used in this work, as many others, is not easily amenable to such exploitation. The code was implemented without initially considering Jacobian sparsity, and modifying it requires nontrivial formulation changes. On the other hand, exploiting sparsity via ADIC2 and ADOL-C is as easy as changing drivers and using different runtime libraries during code compilation. When the Jacobian is sparse, ADIC2 and ADOL-C sparse Jacobians are faster than using the dense ND approach.

Whenever AD is faster than ND, probably no other differentiation method can generate more accurate and more efficient Jacobian matrices with such a short implementation time. Techniques like MD, which could provide machine-precision and efficient derivatives, would be nearly infeasible for complex formulations like the one considered here, and definitely not as general. Nevertheless, there is still room for the improvement of AD performance.

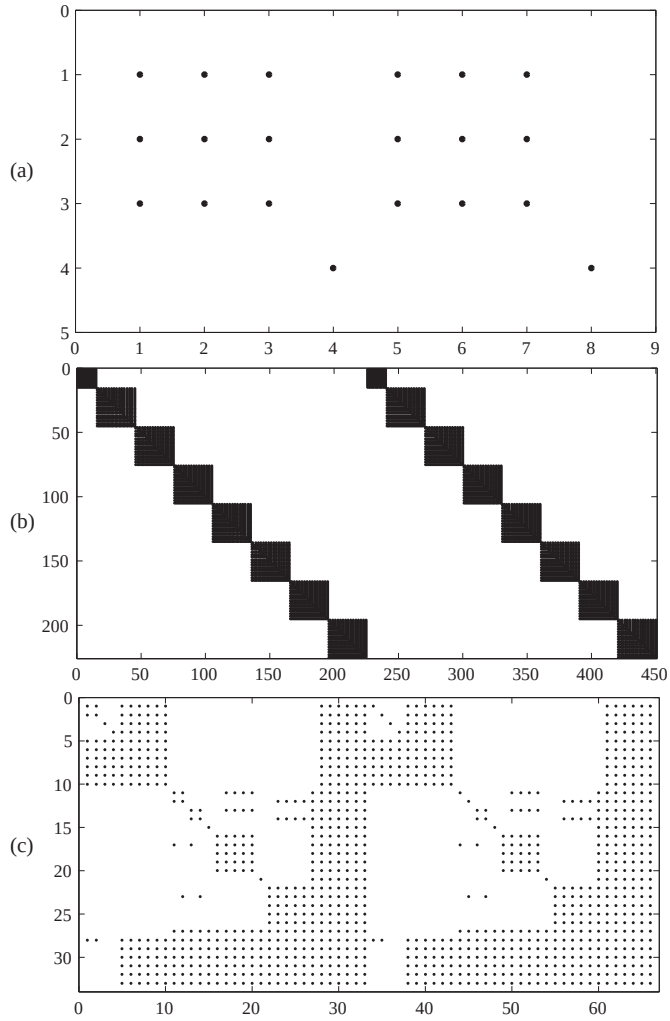


Figure 4.13: Sparsity pattern of (a) the spatial four-bar example, (b) the 7×15 multiple four-bar linkage, and (c) the coach.

Considering the problem as a whole, the results suggest that AD cannot be accepted or rejected over ND by considering just the computational efficiency of the code. Several aspects such as ease of implementation, development time, accuracy, Jacobian structure, and generality should also be assessed. Overall, advantages and disadvantages of state-of-the-art tools for the AD of C/C++ codes in the field of multibody dynamics have been presented rigorously.

From a practical point of view, these results show that ADOL-C and ADIC2 have very similar performances and capabilities. Since ADIC2 is in an earlier stage of development and implies larger implementation challenges (especially for an intermediate programmer), ADOL-C has been chosen to implement the sensitivity analyses in Chapter 5. However, the development of a source-to-source code for the sensitivity analysis is not ruled out, and would certainly be interesting for future works and comparisons.

(a) Spatial four-bar mechanism

h (ms)	Platform	Elapsed Time (s)						Energy Drift (%)	# Jacobians
		ND	ADOL-C			ADIC2			
			Forward	Reverse	Sparse	Forward	Sparse		
1	gcc-1	0.38	0.63	0.77	0.49	0.44	0.42	+0.413	5000
	gcc-2	0.36	0.62	0.91	0.54	0.32	0.30		
10	gcc-1	0.04	0.06	0.80	0.04	0.04	0.04	+0.345	500
	gcc-2	0.05	0.08	0.09	0.06	0.04	0.05		
20	gcc-1	0.02	0.30	0.30	0.04	0.02	0.02	–0.666	250
	gcc-2	0.03	0.05	0.05	0.03	0.03	0.03		

(b) Multiple four-bar linkages

Case	h (ms)	Platform	Elapsed Time (s)						Energy Drift (%)	# Jacobians
			ND	ADOL-C			ADIC2			
				Forward	Reverse	Sparse	Forward	Sparse		
1×15	1	gcc-1	44.80	82.13	49.51	51.17	69.03	51.85	–0.000	5000
		gcc-2	36.49	59.23	44.17	39.45	55.51	37.71		
	10	gcc-1	5.12	8.45	5.04	5.36	7.20	6.32	–0.015	500
		gcc-2	3.78	6.05	4.59	4.14	5.79	4.20		
	20	gcc-1	2.93	4.76	2.98	3.12	4.12	3.39	–0.057	287
		gcc-2	2.23	3.50	2.70	2.40	3.71	2.51		
4×15	1	gcc-1	486.23	1,064.92	889.21	284.73	922.82	309.83	+0.000	5000
		gcc-2	453.45	673.61	538.85	205.57	665.96	214.49		
	10	gcc-1	58.16	106.19	92.77	34.27	97.43	39.09	–0.015	500
		gcc-2	50.38	72.86	58.19	25.26	71.64	28.27		
	20	gcc-1	34.58	60.78	53.97	20.33	55.65	23.82	–0.058	279
		gcc-2	29.85	41.33	33.28	15.13	41.16	17.34		
7×15	1	gcc-1	1,870.84	3,688.65	3,041.90	1,147.13	3,468.08	1,103.67	+0.000	5000
		gcc-2	1,682.25	2,025.48	1,909.90	672.06	Infeasible	Infeasible		
	10	gcc-1	243.85	409.49	346.91	168.81	395.48	167.31	–0.015	500
		gcc-2	198.65	249.65	216.26	96.87	258.26	106.26		
	20	gcc-1	146.90	241.64	202.23	102.68	228.39	103.38	–0.058	275
		gcc-2	116.18	145.88	123.16	59.90	149.04	66.38		

(c) Coach dynamic maneuver

h (ms)	Platform	Elapsed Time (s)						Energy Drift (%)	# Jacobians
		ND	ADOL-C			ADIC2			
			Forward	Reverse	Sparse	Forward	Sparse		
1	gcc-1	48.68	75.43	58.96	66.92	55.19	58.20	+3.742	5000
	gcc-2	36.01	57.93	55.14	50.46	54.44	52.88		
10	gcc-1	5.06	7.60	6.20	6.64	5.60	5.94	+3.731	506
	gcc-2	3.69	5.88	5.78	5.18	5.74	5.41		
20	gcc-1	2.70	4.04	3.19	3.51	3.01	3.26	+3.808	268
	gcc-2	2.03	3.13	2.99	2.74	2.98	2.91		

Table 4.5: ND, ADOL-C and ADIC2 simulation results.

Chapter 5

Sensitivity analysis in independent coordinates

Design sensitivities are nothing but derivatives of the system responses with respect to the system parameters. The quantification of such variations is useful because it allows the importance of a specific design parameter on the outputs to be assessed. When one is trying to find out what parameters can be modified (and to what extent) to achieve a specific result, this information seems very convenient. Quite naturally, gradient-based optimization algorithms for multibody systems can also make use of this information to get to the optimum design efficiently. In the words of Serban and Haug, 1998, "the dynamic design sensitivity analysis of multibody systems (...) represents the link between optimization tools and modeling and simulation tools".

However good these ideas may seem, the computation of sensitivities in the context of forward multibody dynamics is a dynamic process that faces, among others, the difficulties of numerical discretization and rounding, discontinuities in the objective function and the optimization constraints, and mechanical noise. On the other hand, the requirements that a practical sensitivity analysis method should comply with are: convergence, stability and numerical accuracy; easiness of implementation; and computational efficiency. This chapter presents a novel approach for the computation of sensitivities based on the double-step Maggi's formulation (Chapter 2) and the use of automatic differentiation (Chapter 4).

5.1 Introduction

ND is the most straightforward approach to compute response sensitivities. However, according to the experience of many authors, e.g. Etman, 1997, the numerical errors of the sensitivities obtained by ND can cause instabilities in the optimization process. Also, the supply of a user-defined gradient often saves computational time and guarantees a higher accuracy.

From an optimization perspective, finite difference (ND) approaches have additional drawbacks. For example, the optimization algorithm might reach a design point in which the ND gradient does not provide useful information. This happens when, for a given perturbation size, the function cannot be evaluated, either because it is an infeasible point or because the function does not even return real numbers. Conversely, providing a user-defined gradient guarantees that the derivatives can be evaluated to a higher extent and thus the gradient-based algorithm can progress to the next point.

Numerical reasons aside, sensitivity analyses have traditionally been employed to quantify the dependency of a model output with respect to design parameters, and compute function gradients accurately. At the initial design stages, sensitivities help in choosing design parameters. Then, when the model has already been optimized, they identify the most sensitive parameters at the solution point. For all these reasons, the preference for more accurate sensitivities is not just desirable, but rather a requirement in the case of real-life problems. There is, therefore, a need for an integrated dynamic-sensitivity analysis from which accurate sensitivities can be found.

The procedure of obtaining sensitivities is analogous for objective functions and constraint equations, and thus a generic case is considered. It is now worth recalling the definitions made in Section 1.3. Mathematically, the variations of a generic output Ψ can be expressed in terms of the variations in the design parameters through a vector of *design sensitivities* denoted by $\mathbf{s}$:

$$\delta\Psi = \mathbf{s}^T \delta\mathbf{b} \tag{5.1}$$

which is equal to the gradient of the output:

$$\mathbf{s}^T = \frac{d\Psi}{d\mathbf{b}} = \boldsymbol{\nabla}\Psi(\mathbf{b}) = \Psi_{\mathrm{b}} \tag{5.2}$$

The vector of design sensitivities evaluates the influence of each parameter on the objectives, for a specific design. Gradient-based optimization methods use this information to search for new points that improve the value of the objective function and/or the constraint equations. There are three main ways of computing design sensitivities in the context of dynamic mechanical systems, namely *numerical differentiation* (ND), the *adjoint variable method* (AVM) and the *direct differentiation method* (DDM).

Despite the theoretical simplifications, the AVM is found to be involved for general-purpose formulations and real-life problems. Moreover, it requires a forward sweep where all integration information is stored, and a backward sweep afterwards. The backward sweep requires interpolation if the solution points are different from the forward integration points, which can introduce additional errors. For these reasons, the AVM is recommended only when the number of

design variables is very large when compared to the number of outputs (see, for instance, Etman, 1997). In this chapter, the DDM is used instead. According to Pagalday, 1994, the DDM is far easier to implement in large real-life models likes the ones presented here. Even though it would certainly be interesting to make a full comparison including the AVM, due to space limitations and the reasons just mentioned, only the explanation of the AVM is included here. More information on the AVM can be found at Haug and Arora, 1978, Pagalday, 1994, Etman, 1997, and Schaffer, 2005.

5.2 Numerical differentiation approach

If the analytical gradient of the objective function is not available, state sensitivities can be estimated by differencing via finite-difference formulas (ND). This is, of course, the same approach as the one presented in Eq. (4.1). Three examples of ND formulas (forward, backward and centered) for the computation of design sensitivities with respect to a single design parameter b are:

$$\Psi_b(b_0) \equiv \frac{\partial \Psi}{\partial b}(b_0) = \frac{\Psi(b_0 + \Delta b) - \Psi(b_0)}{\Delta b} + O(\Delta b) \tag{5.3}$$

$$\Psi_b(b_0) \equiv \frac{\partial \Psi}{\partial b}(b_0) = \frac{\Psi(b_0) - \Psi(b_0 - \Delta b)}{\Delta b} + O(\Delta b) \tag{5.4}$$

$$\Psi_b(b_0) \equiv \frac{\partial \Psi}{\partial b}(b_0) = \frac{\Psi(b_0 + \Delta b) - \Psi(b_0 - \Delta b)}{2\Delta b} + O(\Delta b^2) \tag{5.5}$$

This approach is certainly simple because it only requires the original function. It also allows for a direct computation of the *state sensitivities*, which are the time-varying derivatives of the state variables with respect to the parameters. Focusing on the position sensitivities, the ND expressions for a single position z would correspond to:

$$z_b(t, b_0) \equiv \frac{\partial z}{\partial b}(t, b_0) = \frac{z(t, b_0 + \Delta b) - z(t, b_0)}{\Delta b} + O(\Delta b) \tag{5.6}$$

$$z_b(t, b_0) \equiv \frac{\partial z}{\partial b}(t, b_0) = \frac{z(t, b_0) - z(t, b_0 - \Delta b)}{\Delta b} + O(\Delta b) \tag{5.7}$$

$$z_b(t, b_0) \equiv \frac{\partial z}{\partial b}(t, b_0) = \frac{z(t, b_0 + \Delta b) - z(t, b_0 - \Delta b)}{2\Delta b} + O(\Delta b^2) \tag{5.8}$$

As already explained in Section 4.1, the need for a small perturbation size Δb in the computation of ND derivatives implies a loss of accuracy. Moreover, the selection of Δb is completely problem- and parameter-dependent, because each system model is based on different physical magnitudes with very different

scales and units. According to the literature, this is an inherent problem that is difficult to control and impossible to avoid.

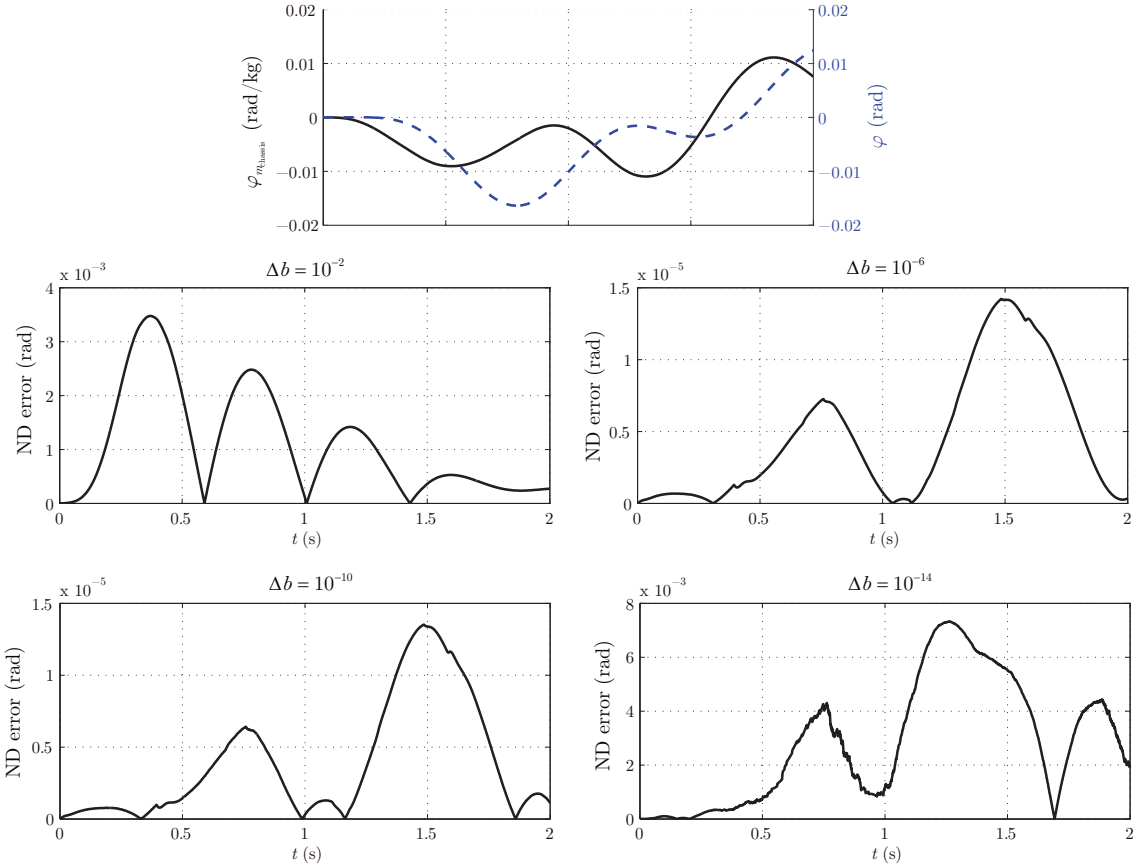


Figure 5.1: Influence of the perturbation size on the accuracy of ND.

In order to assess the influence of the perturbation size Δb on the accuracy of the numerical derivatives, an experiment is conducted. The details of the sample system will be given in the Results section (Section 5.6). Here, the state sensitivity of the bodywork roll angle with respect to the bodywork mass is computed on a simple turn maneuver with a steering angle of:

$$\delta(t) = 0.05 \sin t \, [\text{rad}], \, t \in [0, 2] \tag{5.9}$$

and a velocity of 50 km/h. Figure 5.1 shows five different charts. First, the value of the sensitivity (in black) and the roll angle (in blue) are shown in the top chart. The remaining four charts show the ND absolute error when a centered-difference formula (see Eq. (5.8)) and four different perturbation sizes (10^{-2}, 10^{-6}, 10^{-10} and 10^{-14}) are used. As clearly shown by the figures, the inaccuracy of the state sensitivities can easily grow if the perturbation size is not optimal. This, in turn, has a great influence on the accuracy of the design sensitivities, that is, on the objective function gradient. In order to get a sense of the magnitude of the error, an objective function equal to:

$$\Psi = \sqrt{\int_{t=0}^{t=2} \ddot{z}(t)^2} \tag{5.10}$$

is defined, which corresponds to the RMS value of the vertical acceleration of the chassis. The gradient of this function with respect to 35 coach parameters (described later in Section 5.6.3) is computed by using Eq. (5.5) with the aforementioned perturbation sizes. The gradient is also computed using AD as an advance of what will be explained later. Table 5.1 shows the AD values (accurate to machine precision), the ND values, and the norm of the ND error at the bottom, computed as $\text{error}_{\text{ND}}=\text{norm}(\boldsymbol{\nabla}\Psi_{\text{AD}} - \boldsymbol{\nabla}\Psi_{\text{ND}})$.

From these tests, one can conclude that the error in the ND approach is difficult to predict, and thus the determination of the optimum perturbation value is not straightforward at all. In conclusion, even if a different perturbation size is used for each parameter type, other approaches for the computation of sensitivities ought to be investigated.

5.3 Direct differentiation method

Krishnaswami and Bhatti, 1984, and Chang and Nikravesh, 1985, presented the DDM as a conceptually simpler alternative to the AVM, based on the direct differentiation technique presented by Tomovic, 1963. This method starts by linearizing the objective function and the constraint equations in order to obtain the expressions of the global sensitivities. Let us consider the linearization of a general integral-type objective or constraint function (see Eq. (1.24)):

$$\delta\Psi = \delta\left[\int_{t_i}^{t_f} f(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b})\, dt\right] = \int_{t_i}^{t_f} (f_{\mathbf{z}}\, \delta\mathbf{z} + f_{\dot{\mathbf{z}}}\, \delta\dot{\mathbf{z}} + f_{\ddot{\mathbf{z}}}\, \delta\ddot{\mathbf{z}} + f_{\mathbf{b}}\, \delta\mathbf{b})dt \tag{5.11}$$

where the time variable does not appear in the equation because the integration limits and the integration points do not depend on the design variables. The direct differentiation approach states that the variations in the states can be expressed in terms of the variations in the design, using the Taylor series expansion around point $\mathbf{b}_0$ as follows:

$$\delta\mathbf{z} = \mathbf{z}(t, \mathbf{b}_0 + \delta\mathbf{b}) - \mathbf{z}(t, \mathbf{b}_0) = \mathbf{z}_{\mathbf{b}}\delta\mathbf{b} \tag{5.12}$$

and, similarly, the variations in velocity and acceleration states are:

$$\delta\dot{\mathbf{z}} = \dot{\mathbf{z}}_{\mathbf{b}}\delta\mathbf{b} \tag{5.13}$$

$$\delta\ddot{\mathbf{z}} = \ddot{\mathbf{z}}_{\mathbf{b}}\delta\mathbf{b} \tag{5.14}$$

AD	ND			
	$\Delta b = 10^{-2}$	$\Delta b = 10^{-6}$	$\Delta b = 10^{-10}$	$\Delta b = 10^{-14}$
9.514×10^{-2}	9.517×10^{-2}	9.494×10^{-2}	9.502×10^{-2}	$1.754\times10^{+0}$
-3.668×10^{-1}	-3.733×10^{-1}	-3.667×10^{-1}	-3.669×10^{-1}	$-1.368\times10^{+0}$
-1.938×10^{-5}	-1.483×10^{-5}	-4.611×10^{-6}	-8.320×10^{-5}	5.967×10^{-1}
1.227×10^{-2}	1.229×10^{-2}	1.230×10^{-2}	1.230×10^{-2}	$1.046\times10^{+0}$
1.426×10^{-2}	1.426×10^{-2}	1.426×10^{-2}	1.441×10^{-2}	$-1.124\times10^{+0}$
-3.689×10^{-3}	-3.665×10^{-3}	-3.675×10^{-3}	-3.375×10^{-3}	6.176×10^{-2}
-3.632×10^{-5}	-3.624×10^{-5}	-3.631×10^{-5}	4.580×10^{-5}	-4.649×10^{-1}
4.242×10^{-5}	4.250×10^{-5}	4.250×10^{-5}	1.817×10^{-4}	$1.443\times10^{+0}$
4.139×10^{-6}	4.116×10^{-6}	4.129×10^{-6}	6.779×10^{-5}	$1.343\times10^{+0}$
-9.585×10^{-2}	-9.584×10^{-2}	-9.588×10^{-2}	-9.509×10^{-2}	3.553×10^{-1}
-6.162×10^{-2}	-6.058×10^{-2}	-6.158×10^{-2}	-6.178×10^{-2}	-5.190×10^{-1}
-2.701×10^{-3}	-2.699×10^{-3}	-2.699×10^{-3}	-2.614×10^{-3}	4.316×10^{-1}
-3.513×10^{-4}	-3.512×10^{-4}	-3.512×10^{-4}	-2.331×10^{-4}	$-1.420\times10^{+0}$
-2.897×10^{-4}	-2.897×10^{-4}	-2.897×10^{-4}	-2.690×10^{-4}	1.089×10^{-1}
-2.180×10^{-1}	-2.181×10^{-1}	-2.176×10^{-1}	-2.176×10^{-1}	-2.637×10^{-1}
3.737×10^{-1}	3.739×10^{-1}	3.739×10^{-1}	3.735×10^{-1}	5.378×10^{-1}
-9.444×10^{-1}	-9.443×10^{-1}	-9.448×10^{-1}	-9.450×10^{-1}	$-1.719\times10^{+0}$
1.825×10^{-1}	1.827×10^{-1}	1.827×10^{-1}	1.828×10^{-1}	7.022×10^{-1}
5.041×10^{-1}	5.042×10^{-1}	5.043×10^{-1}	5.042×10^{-1}	$1.799\times10^{+0}$
$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$
$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$	$0.000\times10^{+0}$
-2.273×10^{-1}	-2.274×10^{-1}	-2.274×10^{-1}	-2.274×10^{-1}	$-1.008\times10^{+0}$
-8.738×10^{-2}	-8.734×10^{-2}	-8.738×10^{-2}	-8.799×10^{-2}	-3.921×10^{-1}
-1.486×10^{-5}	-1.479×10^{-5}	-1.478×10^{-5}	-7.286×10^{-5}	$1.308\times10^{+0}$
-2.956×10^{-5}	-2.961×10^{-5}	-2.963×10^{-5}	-1.692×10^{-4}	7.147×10^{-2}
1.443×10^{-5}	1.437×10^{-5}	1.438×10^{-5}	-9.937×10^{-5}	$1.319\times10^{+0}$
5.138×10^{-6}	5.278×10^{-6}	5.217×10^{-6}	9.062×10^{-5}	-4.927×10^{-1}
1.633×10^{-5}	1.638×10^{-5}	1.640×10^{-5}	5.697×10^{-5}	3.053×10^{-1}
-5.567×10^{-6}	-5.706×10^{-6}	-5.699×10^{-6}	-8.812×10^{-5}	8.889×10^{-1}
-8.150×10^{-3}	-8.313×10^{-3}	-8.251×10^{-3}	-8.775×10^{-3}	9.576×10^{-2}
-1.046×10^{-3}	-1.064×10^{-3}	-1.066×10^{-3}	-1.189×10^{-3}	-7.001×10^{-1}
-8.504×10^{-2}	-8.512×10^{-2}	-8.516×10^{-2}	-8.514×10^{-2}	-7.175×10^{-1}
-6.155×10^{-4}	-7.378×10^{-4}	-7.480×10^{-4}	-9.189×10^{-4}	$-1.525\times10^{+0}$
-2.508×10^{-3}	-2.527×10^{-3}	-2.527×10^{-3}	-2.649×10^{-3}	$-1.912\times10^{+0}$
-8.155×10^{-2}	-8.159×10^{-2}	-8.161×10^{-2}	-8.154×10^{-2}	-4.413×10^{-1}
$\text{error}_{\text{ND}} \rightarrow$	6.621×10^{-3}	7.067×10^{-4}	1.603×10^{-3}	$5.298\times10^{+0}$

Table 5.1: ND inaccuracies in the computation of a sample gradient.

Using these expansions, Eq. (5.11) can be rewritten as:

$$\delta\Psi = \int_{t_i}^{t_f} (f_{\mathbf{z}}\mathbf{z}_{\mathbf{b}}\delta\mathbf{b} + f_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}}\delta\mathbf{b} + f_{\ddot{\mathbf{z}}}\ddot{\mathbf{z}}_{\mathbf{b}}\delta\mathbf{b} + f_{\mathbf{b}}\,\mathbf{b})dt \tag{5.15}$$

which is equal to:

$$\delta\Psi = \left[\int_{t_i}^{t_f} (f_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + f_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} + f_{\ddot{\mathbf{z}}}\ddot{\mathbf{z}}_{\mathbf{b}} + f_{\mathbf{b}})dt\right]\delta\mathbf{b} \tag{5.16}$$

and therefore:

$$\mathbf{s}^{\text{int}} = \int_{t_i}^{t_f} (f_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + f_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} + f_{\ddot{\mathbf{z}}}\ddot{\mathbf{z}}_{\mathbf{b}} + f_{\mathbf{b}})dt \tag{5.17}$$

which is the vector of integral global sensitivities. Applying the same procedure to point-type objective functions or constraints (see Eq. (1.10)):

$$\mathbf{s}^{\text{point}} = \left(f_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + f_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} + f_{\ddot{\mathbf{z}}}\ddot{\mathbf{z}}_{\mathbf{b}} + f_{\mathbf{b}}\right)_{t=t_{\max}} \tag{5.18}$$

In both cases, as long as the vectors of *state sensitivities* $\mathbf{z}_{\mathbf{b}}$, $\dot{\mathbf{z}}_{\mathbf{b}}$ and $\ddot{\mathbf{z}}_{\mathbf{b}}$ are available, the global sensitivities $\mathbf{s}$ can be calculated. Thus, the crux of the DDM is to compute state sensitivities rather than global sensitivities.

State sensitivities can be computed by applying the variation principle to the equations of motion in relative coordinates from Maggi's approach, which have been previously presented in Eq. (2.57). The first variation of the motion differential equations would be:

$$(\hat{\mathbf{M}}\ddot{\mathbf{z}}^i)_{\mathbf{z}}\delta\mathbf{z} + (\hat{\mathbf{M}}\ddot{\mathbf{z}}^i)_{\mathbf{b}}\delta\mathbf{b} = (\hat{\mathbf{Q}}_{\mathbf{z}}\delta\mathbf{z} + \hat{\mathbf{Q}}_{\dot{\mathbf{z}}}\delta\dot{\mathbf{z}} + \hat{\mathbf{Q}}_{\mathbf{b}}\delta\mathbf{b}) - (\hat{\mathbf{P}}_{\mathbf{z}}\delta\mathbf{z} + \hat{\mathbf{P}}_{\dot{\mathbf{z}}}\delta\dot{\mathbf{z}} + \hat{\mathbf{P}}_{\mathbf{b}}\delta\mathbf{b}) \tag{5.19}$$

Substituting Eqs. (5.12)—(5.14) into this equation, as done before, results in:

$$(\hat{\mathbf{M}}\ddot{\mathbf{z}}^i)_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + (\hat{\mathbf{M}}\ddot{\mathbf{z}}^i)_{\mathbf{b}} = (\hat{\mathbf{Q}}_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + \hat{\mathbf{Q}}_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} + \hat{\mathbf{Q}}_{\mathbf{b}}) - (\hat{\mathbf{P}}_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + \hat{\mathbf{P}}_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} + \hat{\mathbf{P}}_{\mathbf{b}}) \tag{5.20}$$

Expanding the derivative terms and reordering:

$$\hat{\mathbf{M}}\ddot{\mathbf{z}}^i_{\mathbf{b}} = \hat{\mathbf{Q}}_{\mathbf{b}} + \hat{\mathbf{Q}}_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} + \hat{\mathbf{Q}}_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} - \hat{\mathbf{P}}_{\mathbf{b}} - \hat{\mathbf{P}}_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} - \hat{\mathbf{P}}_{\dot{\mathbf{z}}}\dot{\mathbf{z}}_{\mathbf{b}} - \hat{\mathbf{M}}_{\mathbf{b}}\ddot{\mathbf{z}}^i - (\hat{\mathbf{M}}\underline{\ddot{\mathbf{z}}}^i)_{\mathbf{z}}\mathbf{z}_{\mathbf{b}} \tag{5.21}$$

where subscripts denote partial differentiation and the notation of a bar under a term (_) indicates that it is treated as constant in the partial differentiation.

Equation (5.21) constitutes n_{dp} sets of f second-order ODEs in terms of state sensitivities $\ddot{\mathbf{z}}^i_{\mathbf{b}}$, in contrast with other approaches in the literature where the index of the resulting DAE systems can cause difficulties in the integration. These ODE sets have to be integrated together with the forward dynamics equations (2.57) because the calculation of sensitivities is based on the forward integration of the equations of motion. Interestingly enough, systems (5.21) and (2.57) have the same system matrix $\hat{\mathbf{M}}$, which means the factorization of $\hat{\mathbf{M}}$ can be reused in the computation of $\ddot{\mathbf{z}}^i_{\mathbf{b}}$, as suggested by Pagalday, 1994. Both the AVM and the DDM are considered by Sonneville and Brüls, 2013, to be

semi-analytical methods, as they discretize the time but also require the manual differentiation of many dynamic terms.

Beware that the integration of the system of ODEs in Eq. (5.21) requires an appropriate value of the initial state sensitivities $\mathbf{z}_{\mathrm{b}}$ and $\dot{\mathbf{z}}_{\mathrm{b}}^{i}$. It is the responsibility of the designer to quantify the dependency of the initial states on the design parameters. This can sometimes constitute a problem on itself, as several authors point out (for instance Chang and Nikravesh, 1985). In the present applications, because of the relative nature of the coordinates, the parameter types and the dynamic maneuvers, the initial sensitivities will always be null.

A difference between the AVM and the DDM can now be pointed out. The AVM, as will be shown next, leads to as many additional systems of DAEs to be integrated backwards in time as objective functions (in case there are more than one), regardless of the number of design parameters. In turn, the DDM leads to as many additional systems of DAEs as design parameters, regardless of the number of objective functions. The higher the number of objective functions and constraints, the more advantageous the DDM becomes. Finally, a further advantage of the DDM with respect to the AVM is that it is easier to parallelize, according to the experience of Schaffer, 2005.

The specific form of the sensitivity equations depends on the form of the equations of motion. Even though many authors have applied the DDM to the index-3 Eqs. (1.1), the approach presented here, that is, the differentiation of a minimal set of ODEs (very often computed through the matrix-R method), has hardly ever been tackled before. A light reference is found in García de Jalón and Bayo, 1994, and a more thorough analysis is found in Wang, Haug and Pan, 2005. Later, an introduction to the ideas of this chapter was presented in Gutiérrez-López, Callejo and García de Jalón, 2012. The computation of independent sensitivities from a state-space ODE version of the equations of motion, has the same advantages as the corresponding formulation for the time integration of forward dynamics, as will be proven later. This approach, together with an efficient version of Maggi's equations, is followed in the rest of this work. To the author's knowledge, the only pseudo-real-life example to which the computation of independent sensitivities has been applied in the literature can be found in Wang, Haug and Pan, 2005.

Regardless of the specific equations of motion that are differentiated, the main problem of the DDM is the calculation of the r.h.s. of Eq. (5.21). There are three basic ways in which these terms can be computed: using MD, ND or AD. In the following section, the AD approach is analyzed. To the author's knowledge, the AD approach has not been used before in the context of independent sensitivity analysis of multibody systems. MD is only used for comparison purposes in two of the three examples presented at the end of the chapter.

Finally, note that the ND of Eq. (5.21) would share many of the drawbacks of the ND approach presented in Section 5.2, and is thus not considered here.

5.4 Adjoint variable method

For the purpose of theoretical comparisons, a brief summary of the AVM is presented, even though it has not been implemented due to time constraints. For additional details, see the following references: Haug and Arora, 1978; Chang and Nikravesh, 1985; Etman, 1997; and Schaffer, 2005.

Let us consider a position-, velocity- and parameter-dependent objective function (or optimization constraint), Ψ, of integral type, written in terms of the state vector $\mathbf{y}$ defined in Eq. (2.59):

$$\Psi = \int_{t_i}^{t_f} f(t, \mathbf{y}, \mathbf{b})\, dt \tag{5.22}$$

The variation of such an objective function when a small change in the parameters is introduced can be expressed as:

$$\delta\Psi = \int_{t_i}^{t_f} (f_{\mathrm{y}}\delta\mathbf{y} + f_{\mathrm{b}}\,\delta\mathbf{b})\, dt \tag{5.23}$$

The idea behind the AVM is to eliminate the dependence of the objective function with the states, thus only considering the dependency on the parameters. To that end, Eq. (2.59) is multiplied by a vector of *adjoint variables*, $\boldsymbol{\mu}$, and is integrated over time as follows:

$$\int_{t_i}^{t_f} \boldsymbol{\mu}^T(\dot{\mathbf{y}} - \mathbf{f})\, dt = 0 \tag{5.24}$$

where the adjoint variable function $\boldsymbol{\mu} = \boldsymbol{\mu}(t)$ can take any form. The variation of the previous integral when a small change in the parameters is introduced can be expressed as:

$$\int_{t_i}^{t_f} \boldsymbol{\mu}^T \left(\delta\dot{\mathbf{y}} + \dot{\mathbf{y}}_{\mathrm{b}}\delta\mathbf{b} - \mathbf{f}_{\mathrm{y}}\delta\mathbf{y} - \mathbf{f}_{\mathrm{b}}\delta\mathbf{b}\right) dt = 0 \tag{5.25}$$

Integrating the first term by parts, the following expression holds:

$$\int_{t_i}^{t_f} \boldsymbol{\mu}^T \delta\dot{\mathbf{y}}\, dt = \underbrace{\boldsymbol{\mu}^T \delta\mathbf{y}\Big|_{t_i}^{t_f}}_{0} - \int_{t_i}^{t_f} \delta\mathbf{y}\, \dot{\boldsymbol{\mu}}^T dt \tag{5.26}$$

where the first term of the r.h.s. is null if we assume that the initial configuration does not depend on the design parameters and that $\boldsymbol{\mu}(t_f) = \mathbf{0}$. Introducing Eq. (5.26) in Eq. (5.25) and reordering:

$$-\int_{t_i}^{t_f} \left(\dot{\boldsymbol{\mu}}^T + \boldsymbol{\mu}^T \mathbf{f}_{\mathrm{y}}\right) \delta\mathbf{y}\, dt = \int_{t_i}^{t_f} \boldsymbol{\mu}^T \left(-\dot{\mathbf{y}}_{\mathrm{b}}\delta\mathbf{b} + \mathbf{f}_{\mathrm{b}}\delta\mathbf{b}\right) dt \tag{5.27}$$

The vector of adjoint variables can be such that:

$$\dot{\boldsymbol{\mu}}^T + \boldsymbol{\mu}^T \mathbf{f}_{\mathrm{y}} = f_{\mathrm{y}} \tag{5.28}$$

In that case, Eq. (5.27) can be rewritten as:

$$\int_{t_i}^{t_f} f_{\mathrm{y}} \delta \mathbf{y}\, dt = \int_{t_i}^{t_f} \boldsymbol{\mu}^T \left(\dot{\mathbf{y}}_{\mathrm{b}} \delta \mathbf{b} - \mathbf{f}_{\mathrm{b}} \delta \mathbf{b} \right) dt \tag{5.29}$$

and introducing this equation into Eq. (5.23) leads to:

$$\delta \Psi = \int_{t_i}^{t_f} \left[\boldsymbol{\mu}^T \left(\dot{\mathbf{y}}_{\mathrm{b}} \delta \mathbf{b} - \mathbf{f}_{\mathrm{b}} \delta \mathbf{b} \right) + f_{\mathrm{b}}\, \delta \mathbf{b} \right] dt \tag{5.30}$$

which can be rearranged into:

$$\mathbf{s}^{\mathrm{int}} \equiv \frac{\delta \Psi}{\delta \mathbf{b}} = \int_{t_i}^{t_f} \left[\boldsymbol{\mu}^T \left(\dot{\mathbf{y}}_{\mathrm{b}} - \mathbf{f}_{\mathrm{b}} \right) + f_{\mathrm{b}} \right] dt \tag{5.31}$$

This equation allows computing the design sensitivities of the integral-type objective function. The procedure for its solution involves several steps. First, the equations of motion are integrated from t_i to t_f via Eq. (2.59) (as was comprehensively described in Chapter 2), and state vectors $\mathbf{y}$ and state vector derivatives $\dot{\mathbf{y}}$ are stored. Second, the vector of adjoint variables $\boldsymbol{\mu}(t)$ is integrated from t_f to t_i using Eq. (5.28) with $\boldsymbol{\mu}(t_f) = \mathbf{0}$ as initial conditions. Third, and last, Eq. (5.31) is solved for the sensitivities. Step 1 has to be solved only once, while steps 2 and 3 have to be solved $(n_{\mathrm{of}} + n_{\mathrm{oc}})$ times.

The AVM presented here, even though rather complicated, is one of its simplest forms, and does not account for the dependency of the objective function on $\dot{\mathbf{y}}$. That kind of dependency would require additional sets of adjoint variables, making the method even more cumbersome. Moreover, care must be taken not to incur numerical errors in the algorithm, especially when the forward integration points do not coincide with the backward ones and interpolation is required. Even though the AVM could be handy for problems with lots of design variables, all these reasons make it somewhat impractical for a general-purpose sensitivity analysis code.

5.5 Automatic differentiation approach

The approach for the computation of design and state sensitivities developed in this Thesis combines the DDM (explained in Section 5.3) and the AD technique (presented in Chapter 4) into a hybrid direct-automatic differentiation method. Specifically, the operator overloading tool ADOL-C (Griewank, Juedes and Utke, 1996) is used to compute algorithmically all the terms that would otherwise need to be computed manually.

5.5.1 Tape generation

The function that computes the state vector derivative has three types of input variables: the design parameters $\mathbf{b}$, the state vector $\mathbf{y}$ and the time variable t. In terms of these variables, the motion differential equations and the sensitivity equations can be written as:

$$\dot{\mathbf{y}} = \dot{\mathbf{y}}(\mathbf{b}, \mathbf{y}, t) \tag{5.32}$$

$$\dot{\mathbf{y}}_{\mathrm{b}} = \dot{\mathbf{y}}_{\mathrm{b}}(\mathbf{b}, \mathbf{y}, \mathbf{y}_{\mathrm{b}}, t) \tag{5.33}$$

$$\mathbf{y}^T \equiv \{\mathbf{z}^T, \dot{\mathbf{z}}^{iT}\} \tag{5.34}$$

Let us consider the function under differentiation, $\dot{\mathbf{y}}$. Even though all inputs affect the output, sensitivity analyses are only concerned with derivatives with respect to $\mathbf{b}$, namely $\dot{\mathbf{y}}_{\mathrm{b}}$. Thus, it would be desirable to distinguish between active inputs with respect to which the function will be differentiated, and passive inputs which affect the result but whose derivatives are not desired. However, ADOL-C, in its current state, forces the user to define all varying variables as AD inputs, and whenever the derivatives are evaluated, sensitivities with respect to all inputs are returned. For this reason, an input variable vector $\mathbf{x}$ grouping the three arrays is introduced. The function $\dot{\mathbf{y}}(\mathbf{x})$ and its derivative with respect to $\mathbf{b}$ can therefore be written as:

$$\dot{\mathbf{y}} = \dot{\mathbf{y}}(\mathbf{x}) \tag{5.35}$$

$$\begin{aligned} \dot{\mathbf{y}}_{\mathrm{b}} = \dot{\mathbf{y}}_{\mathrm{x}}\mathbf{x}_{\mathrm{b}} &= \begin{bmatrix} \dfrac{\partial \dot{\mathbf{y}}}{\partial \mathbf{b}} & \dfrac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} & \dfrac{\partial \dot{\mathbf{y}}}{\partial t} \end{bmatrix} \begin{bmatrix} \dfrac{d\mathbf{b}}{d\mathbf{b}} \\ \dfrac{d\mathbf{y}}{d\mathbf{b}} \\ \dfrac{dt}{d\mathbf{b}} \end{bmatrix} = \\ &= \begin{bmatrix} \dfrac{\partial \dot{\mathbf{y}}}{\partial \mathbf{b}} & \dfrac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} & \dfrac{\partial \dot{\mathbf{y}}}{\partial t} \end{bmatrix} \begin{bmatrix} \mathbf{I}_{n_{\mathrm{dp}}} \\ \dfrac{d\mathbf{y}}{d\mathbf{b}} \\ 0 \end{bmatrix} = \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{b}} + \frac{\partial \dot{\mathbf{y}}}{\partial \mathbf{y}} \frac{d\mathbf{y}}{d\mathbf{b}} \end{aligned} \tag{5.36}$$

$$\mathbf{x}^T \equiv \{\mathbf{b}^T, \mathbf{y}^T, t\} \tag{5.37}$$

Once function $\dot{\mathbf{y}}(\mathbf{x})$ has been taped by ADOL-C, both the function and the derivative of the function with respect to the inputs, $\dot{\mathbf{y}}_{\mathrm{x}}$, can be evaluated using ADOL-C drivers. Then, using Eq. (5.36), the Jacobian matrix of the outputs with respect to the parameters, $\dot{\mathbf{y}}_{\mathrm{b}}$, can be calculated and thus integrated to obtain sensitivities $\mathbf{y}_{\mathrm{b}}$ over time (along with state vectors $\mathbf{y}$).

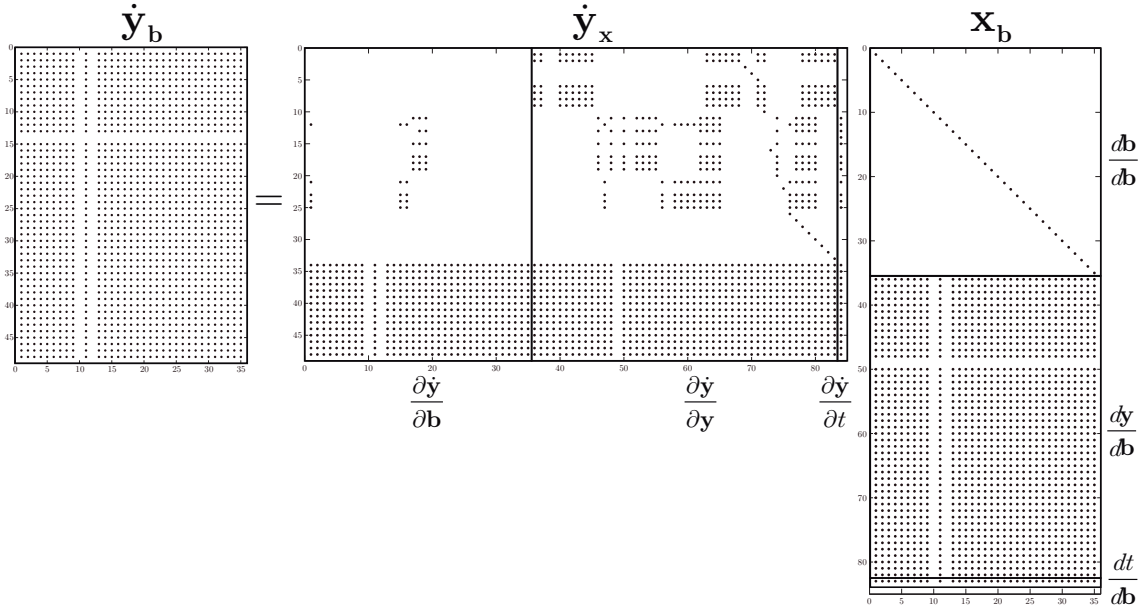


Figure 5.2: Sparsity patterns of the state vector Jacobian matrix components.

As far as the sparse nature of the matrices is concerned, Figure 5.2 shows the sparsity pattern of matrices $\dot{\mathbf{y}}_\mathbf{b}$ and $\dot{\mathbf{y}}_\mathbf{x}$, where the black dots represent the nonzeroes of the matrices. In particular, matrices corresponding to the coach model from Chapter 3 with 35 design parameters are shown. More details on the types of sensitivities will of course be provided in the following section. The sparsity patterns are kept during the whole integration process.

5.5.2 Program structure

There are several ways in which the code can be organized to compute state sensitivities. Let us apply the Leibniz rule for differentiation of integrals to the sensitivity equations (5.21). The differentiation with respect to the design parameters $\mathbf{b}$ can be carried out during or after the time integration of the equations of motion, as Figure 5.3 shows. It is clear how, in every time step, the sensitivity equations have to be solved after solving the forward dynamics. In the level of differentiation ①, AD is used to compute the generic derivatives of function $\mathbf{f}$ with respect to $\mathbf{b}$, namely $\mathbf{f}_\mathbf{b}$. The tape of the differentiated function is then used to evaluate each of the inner integrator calls. On the other hand, in the level of differentiation ②, the results of the integration themselves, $\mathbf{Y} = \left[\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_{\text{end}}\right]^T$, are differentiated with respect to $\mathbf{b}$.

Apart from these possibilities, ADOL-C offers further ways of implementation, with different modes and function tapes, according to which several schemes of differentiation (approaches) can be proposed. Recall, for a better understanding of the following subsections, the basic drivers provided by ADOL-C and briefly explained in Section 4.5.

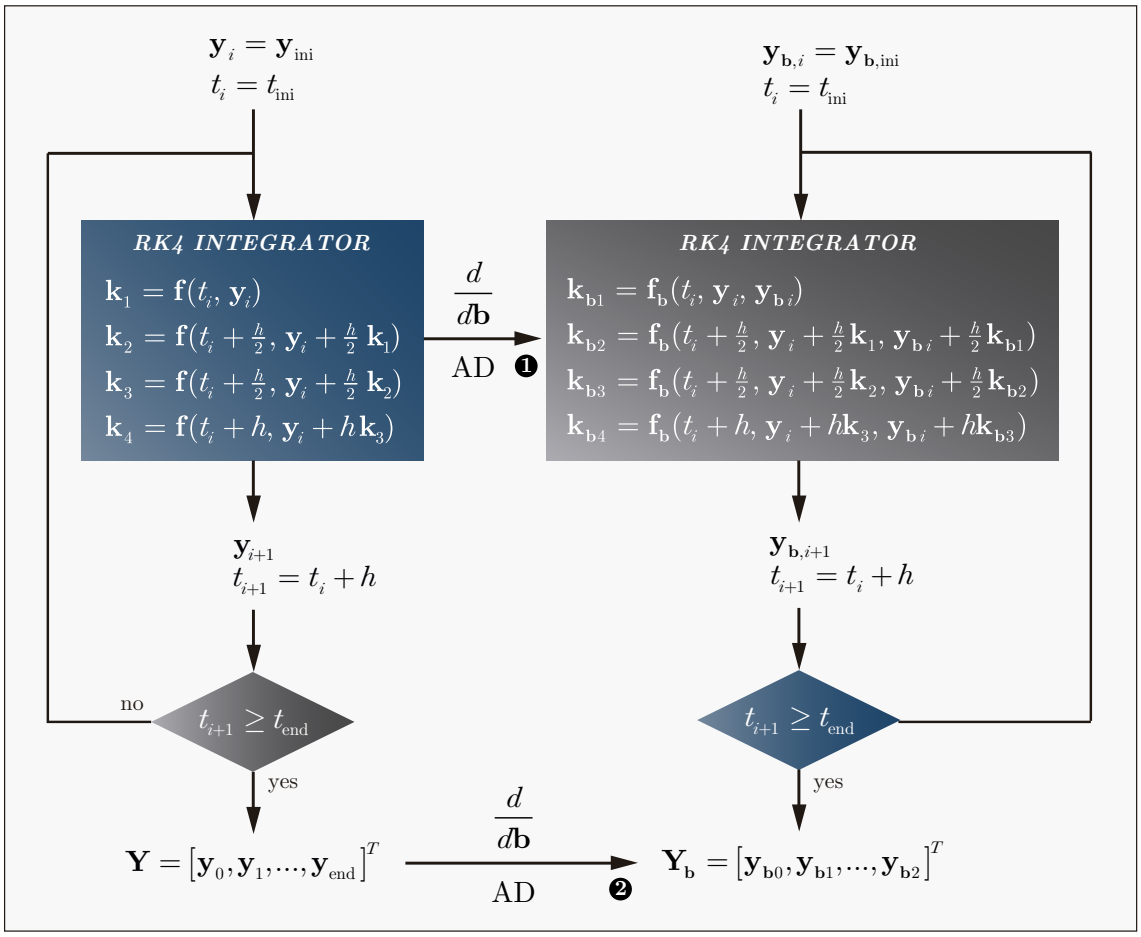


Figure 5.3: Flow of the time integration of dynamic and sensitivity equations.

First approach The first way of implementing AD is based on the level of differentiation ①, as shown in Figure 5.3. It consists of taping the function $\mathbf{f}$ the first time the state vector derivative function is run, and then using the driver `fov_forward` to compute both $\mathbf{f}$ and $\mathbf{f}_{\mathbf{b}}$ simultaneously each of the four times required by the 4^{th} order Runge-Kutta integrator. Let us recall the expression of the state-space sensitivity accelerations from (5.21):

$$\mathbf{f}^{\text{AD}} \equiv \hat{\mathbf{M}}(\mathbf{z}, \mathbf{b})^{-1} \left[\hat{\mathbf{Q}}(t, \mathbf{z}, \dot{\mathbf{z}}, \mathbf{b}) - \hat{\mathbf{P}}(t, \mathbf{z}, \dot{\mathbf{z}}, \mathbf{b}) \right] \tag{5.38}$$

$$\ddot{\mathbf{z}}_{\mathbf{b}}^{i} = \mathbf{f}_{\mathbf{b}}^{\text{AD}}(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{z}_{\mathbf{b}}, \dot{\mathbf{z}}_{\mathbf{b}}, \mathbf{b}) \tag{5.39}$$

In this case, the complete computation of the state vector derivative is recorded to the AD tape, including the solution of the linear systems in Eqs. (2.50), (2.55) and (2.57). The latter is solved by using Cholesky decomposition because the system matrix $\hat{\mathbf{M}}$ is a positive-definite matrix, but the first two systems are usually solved using an LU decomposition.

One of the limitations of this approach is that the LU factorization and its corresponding back-substitution cannot be employed in the generation of the tape, because they imply reordering of the rows of the system matrices based on con-

ditional comparisons, and thus, as explained, cannot be efficiently taped by ADOL-C. To solve this problem, an AD version of the code is written, where the first two linear systems of equations are solved by using the Cholesky factorization instead of the LU decompositions. To do so, the linear systems have to be rewritten so that the system matrix is square. For instance, the upper submatrix $\mathbf{R}_{\mathbf{z}}^{u}$ of Eq. (2.50) would become:

$$\mathbf{\Phi}_{\mathbf{z}}^{d}\,\mathbf{R}_{\mathbf{z}}^{u} = -\mathbf{\Phi}_{\mathbf{z}}^{i} \tag{5.40}$$

$$\mathbf{\Phi}_{\mathbf{z}}^{dT}\,\mathbf{\Phi}_{\mathbf{z}}^{d}\,\mathbf{R}_{\mathbf{z}}^{u} = -\mathbf{\Phi}_{\mathbf{z}}^{dT}\mathbf{\Phi}_{\mathbf{z}}^{i} \tag{5.41}$$

$$\mathbf{R}_{\mathbf{z}}^{u} = -\left(\mathbf{\Phi}_{\mathbf{z}}^{dT}\,\mathbf{\Phi}_{\mathbf{z}}^{d}\right)^{-1}\left(\mathbf{\Phi}_{\mathbf{z}}^{dT}\mathbf{\Phi}_{\mathbf{z}}^{i}\right) \tag{5.42}$$

Likewise, the second system derived from Eq. (2.55). These changes are effective from an AD point of view, but obviously have a computational cost.

Second approach The second way of implementing ADOL-C is also based on the level of differentiation ①, according to Figure 5.3. In this scheme, the factorization of the system matrix $\hat{\mathbf{M}}$ carried out for the dynamic equations is reused for the computation of independent state sensitivities in Eq. (5.21). Therefore, the automatically-differentiated code, $\mathbf{f}^{\mathrm{AD}}$, would not include the solution of the system of linear equations, as the next expressions show:

$$\mathbf{f}^{\mathrm{AD}} \equiv \hat{\mathbf{Q}}(t,\mathbf{z},\dot{\mathbf{z}},\mathbf{b}) - \hat{\mathbf{P}}(t,\mathbf{z},\dot{\mathbf{z}},\mathbf{b}) \tag{5.43}$$

$$\ddot{\mathbf{z}}_{\mathbf{b}}^{i} = \hat{\mathbf{M}}(\mathbf{z},\mathbf{b})^{-1}\mathbf{f}_{\mathbf{b}}^{\mathrm{AD}}(t,\mathbf{z},\dot{\mathbf{z}},\ddot{\mathbf{z}},\mathbf{z}_{\mathbf{b}},\dot{\mathbf{z}}_{\mathbf{b}},\mathbf{b}) \tag{5.44}$$

This approach requires the separate computation of $\hat{\mathbf{M}}$. The dynamic equations can be computed both with the original function and with the `fov_forward` driver of ADOL-C. The solution of the linear system is carried out by using the Cholesky factorization and back-substitution in both the dynamic equations and the sensitivity equations.

Third approach The third scheme is based on the first approach and on the use of ADOL-C's sparse drivers. When the Jacobian matrix is sparse, it is sometimes useful to propagate only the derivatives that correspond to nonzeroes in the Jacobian matrix. ADOL-C provides a driver for the exploitation of sparsity in the computation of Jacobian matrices, called `sparse_jac`. The main disadvantage of this approach is that the `sparse_jac` driver only returns the Jacobian matrix and not the function value, which means it has to be computed separately with a call to the `function` driver.

Fourth approach The fourth scheme is based on the third approach. The only difference is that instead of computing the accelerations (i.e., the dynamic equations) by using ADOL-C's `function` driver, the original code is used. A comparison between approaches three and four can be useful to determine which of the functions is more efficient: the original code or the code based on ADOL-

C tape. In the author's experience, as will be shown later in Section 5.5.3, the computation time for both approaches is very similar, meaning that the original code is already efficient.

Fifth approach Finally, perhaps the most straightforward and conceptually simple way of obtaining the sensitivities is to differentiate the final states altogether. This approach corresponds to the level of differentiation ② in Figure 5.3, i.e., to the differentiation of the integrated equations. From the computational point of view, the final state matrix is just the result of an ensemble of computer operations. From this perspective, the outputs of the computational tree (state matrix $\mathbf{Y}$) can be differentiated with respect to the inputs (design parameters $\mathbf{b}$).

The main drawback of this approach is the huge amount of operations to be recorded on the tape (or, in source-to-source approaches, the size of the generated code). A simulation of a 10-s maneuver with 1-ms time-step would require the recording (and then differentiation) of 40,000 evaluations of the state vector derivative in a row. This constitutes a challenge both from the system memory and efficiency points of view. Some AD tools provide a functionality called *checkpointing*, which allows recording only specific checkpoints during the execution of the original function, enabling a more efficient evaluation of derivatives later. However, this is still an active topic of research and has not been applied to complex multibody codes or examples, and therefore has not been implemented here.

5.5.3 Efficiency

As far as the presented differentiation approaches are concerned, an example has been run in order to discern which of the four has a better performance. The system is the coach model presented in previous chapters. More details will be provided in the Results section.

	AD approach			
	1	2	3	4
Drivers used	`fov_forward`	`fov_forward`	`function sparse_jac`	`sparse_jac`
Independent variables	88	88	88	88
Dependent variables	52	377	52	52
Operations	143078	146311	143078	143078
Locations	370803	381042	370803	370803
Values	35631	35630	35631	35631
Elapsed time (s)	83.07	85.63	195.52	189.49

Table 5.2: Characteristics and computation times of AD schemes.

A sensitivity analysis has been run using each of those schemes, obtaining the computation times shown in Table 5.2. These results show that approach 1 is the fastest one, closely followed by approach 2. Approaches 3 and 4 are much slower and similar to one another, meaning that the difference between evaluating the original function and evaluating the tape is negligible. In the following section, approach number 1 will be used for all numerical examples, because it is the most competitive one.

5.6 Results

Two academic examples and one real-life example are analyzed so as to assess the performance of the presented approach. Note that the presented numerical framework is completely general-purpose and automatic, thus enabling the sensitivity analysis of virtually any multibody system. If the external forces and user functions can be coded, the AD implementation will automatically propagate the derivatives and compute the sensitivities to machine precision.

In addition to ND and AD sensitivities, thanks to the results provided by M. D. Gutiérrez-López, two of the three examples have been solved using analytical (manual) direct differentiation (MD). This provides a useful comparison with a third differentiation technique. More details on this approach can be found at Gutiérrez-López, Callejo and García de Jalón, 2012.

5.6.1 Double-pendulum

The double-pendulum example has two bodies, two parallel revolute joints and two DOFs. The initial configuration can be seen in Figure 5.4(a). Bar 1 has an initial angular velocity of -1 rad/s, whereas the initial relative velocity of joint 2 is null. Both bars have a length of 1 m and a uniformly distributed mass of 1 kg. A spring-damper set is attached to joint 1, and has the following properties: $k = 1$ N·m/rad, $c = 1$ N·m·s/rad. The simulation time is 5 s, with a time-step of 10 ms. Position, velocity and acceleration sensitivities with respect to the following design parameters are computed:

– Mass of bar 1 (m)

– Moment of inertia of bar 1 (I_x)

– Y-coordinate of the COG of bar 1 (y_{COG})

– Damping of joint 1 (c)

– Stiffness of joint 1 (k)

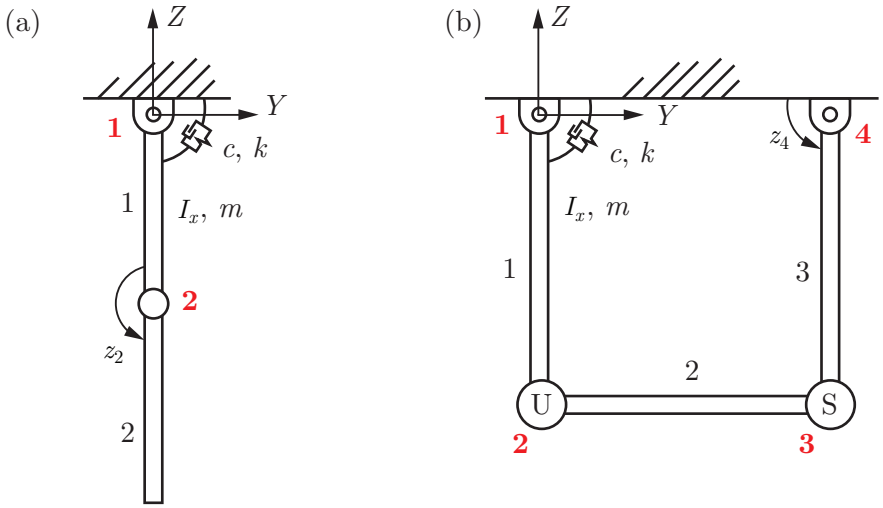


Figure 5.4: (a) Double-pendulum. (b) Four-bar mechanism.

Figure 5.5 shows the sensitivity of z_2 with respect to m and the absolute error of ND. A good agreement between the results obtained by ND, MD and AD is achieved. ND implies a relatively high error which depends on the perturbation size and is not easy to control, whereas MD and AD yield exact sensitivities.

Elapsed time (s)	$\mathbf{b} = \{m, I_x, y_{\text{COG}}, c, k\}^T$		
	ND	MD	AD
Dynamics	0.003	0.003	0.003
Sensitivity	0.051	0.106	0.072
Total	0.055	0.110	0.075
Sensitivity/Total	93%	97%	95%

Table 5.3: Computation times of the 5-second double-pendulum simulation.

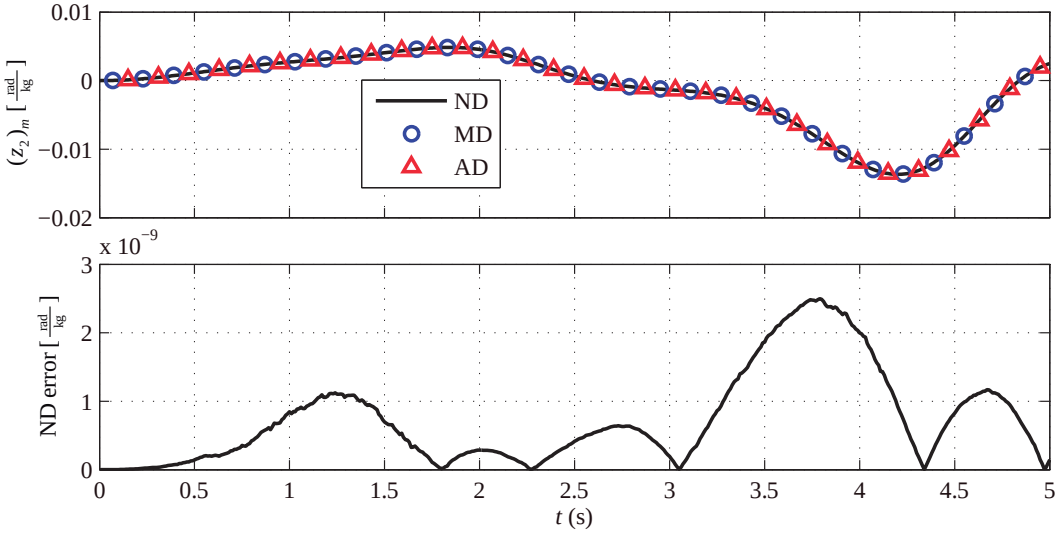


Figure 5.5: Double pendulum: sensitivity of z_2 with respect to m and ND error.

Table 5.3 shows the computation times of the dynamic analysis (first row), the full sensitivity analysis (second row) and the addition of both (third row) with each of the three methods. The last row contains the weight of the sensitivity computation relative to the total time. Only the five-parameter case is shown as a summary. However, in order to assess the effect of the number of design parameters on the efficiency, executions with different numbers of parameters have been run to produce a time vs. # parameters plot (see Figure 5.9). In this specific example, elapsed times are probably too short to draw useful conclusions. Nevertheless, ND and AD seem to be the fastest approaches.

5.6.2 Four-bar mechanism

The four-bar mechanism is a closed-loop system with three moving bodies, four joints (revolute, universal, spherical and revolute) and one DOF. The configuration in the initial position is the one depicted in Figure 5.4(b). Bar 1 has an initial angular velocity of -1 rad/s. All bars have the same properties (length, mass and inertia moments) as the bars of the previous example. Similarly, a spring-damper set with the same properties is attached to joint 1. The simulation time is 5 s, and a 10-millisecond time-step has been used. Again, the following sensitivities are computed:

– Mass of bar 1 (m)

– Moment of inertia of bar 1 (I_x)

– Y-coordinate of the COG of bar 1 (y_{COG})

– Damping of joint 1 (c)

– Stiffness of joint 1 (k)

Elapsed time (s)	$\mathbf{b} = \{m, I_x, y_{\mathrm{COG}}, c, k\}^T$		
	ND	MD	AD
Dynamics	0.009	0.009	0.009
Sensitivity	0.126	0.160	0.145
Total	0.136	0.170	0.155
Sensitivity/Total	93%	94%	94%

Table 5.4: Computation times of the 5-second four-bar mechanism simulation.

Figure 5.6 shows the sensitivity of z_4 with respect to m and the absolute error of ND. A good agreement between ND, MD and AD is achieved. ND entails a numerical error, which gets larger when the model complexity increases. Although the size of the ND perturbation could be scaled differently for each design parameter, the error is very sensitive to the nature of the parameters, and thus it is impractical. Computation times of the full sensitivity analysis (with five

parameters) are shown in Table 5.4, and elapsed times with one, three and five parameters are plotted in Figure 5.9. The trend of the previous example recurs, although MD times are now closer to ND and AD times.

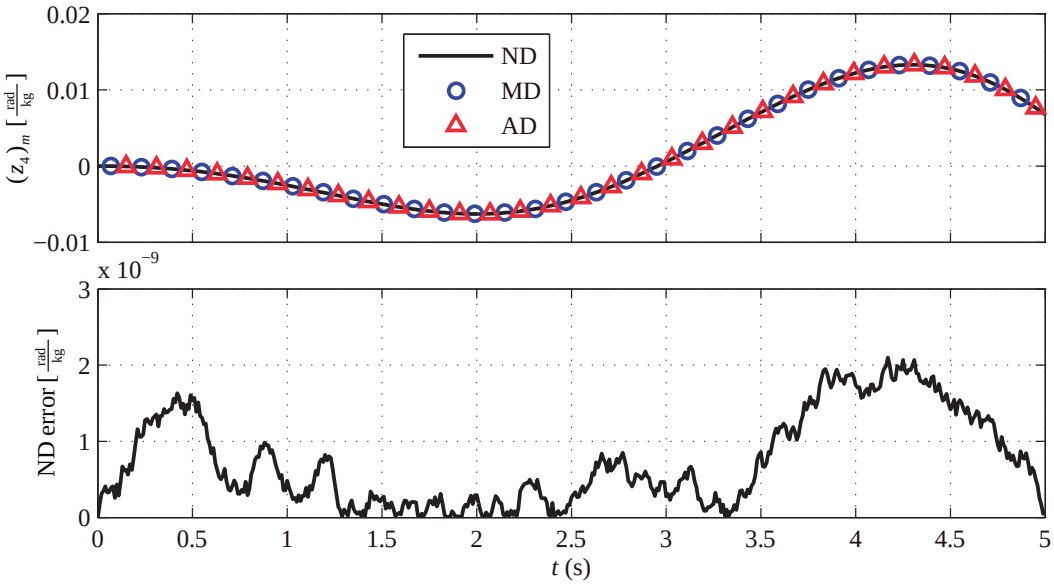


Figure 5.6: Four-bar mechanism: sensitivity of z_4 with respect to m and ND error.

5.6.3 Coach maneuver

Finally, the 18-DOF coach presented in Chapter 3 is analyzed. Specifically, a 2-second single lane-change maneuver has been run with a time-step of 1 ms. The angle coordinate of the steering system is kinematically driven with a predefined steering function $\delta(t) = 0.05 \sin t$ [rad], $t \in [0,2]$. A constant torque of 600 N·m is applied on the rear wheels' joints. Regarding design sensitivities, a set of 35 design parameters has been considered, ranging from inertias to distance parameters, and are listed in Table 5.5. In this example, because of practical limitations with MD, only ND and AD sensitivities have been computed. Sixteen different sensitivities and their corresponding ND errors have been plotted in Figures 5.7 and 5.8. Specifically, the derivatives of z (Z-coordinate of the bodywork) and φ (roll angle of the bodywork), respectively, with respect to eight of the most relevant parameters, have been plotted. As in the previous examples, the agreement between ND and AD is fully satisfactory.

It is not uncommon, when optimizing complex mechanical systems with many design parameters towards a desired behavior, to have hesitations about the *relevance* of specific parameters. Without accurate sensitivity analyses, the designer might be disoriented about the parameter choice until the optimization is run. A wrong choice of design parameters can lead to a waste of time and resources trying to optimize the system response, or even worse, to a mediocre result that could have been better with an optimal choice of parameters.

#	Name	Units
1	*Z*-coordinate of the joint unit vector of the left triangles ($u_{z,\text{triangle}}$)	m
2	Mass of the bodywork (m_{chassis})	kg
3	*XX* product of inertia of the bodywork (I_{xx})	kg·m^2
4	*YY* product of inertia of the bodywork (I_{yy})	kg·m^2
5	*ZZ* product of inertia of the bodywork (I_{zz})	kg·m^2
6	Mass of the rear support	kg
7	*XX* product of inertia of the rear support	kg·m^2
8	*YY* product of inertia of the rear support	kg·m^2
9	*ZZ* product of inertia of the rear support	kg·m^2
10	*X*-position of the COG	m
11	*Y*-position of the COG	m
12	*Z*-position of the COG	m
13	Mass of rod 1 (m_{rod})	kg
14	Mass of rod 2	kg
15	*X*-coordinate of the joint unit vector of the left triangles	m
16	*Y*-coordinate of the joint unit vector of the left triangles	m
17	*X*-coordinate of the joint unit vector of the right triangles	m
18	*Y*-coordinate of the joint unit vector of the right triangles	m
19	*Z*-coordinate of the joint unit vector of the right triangles	m
20	*Y*-distance to rear left wheel joint (y_{track})	m
21	*Y*-distance to rear right wheel joint	m
22	*Y*-distance to front left wheel joint	m
23	*Y*-distance to front right wheel joint	m
24	*XX* product of inertia of rear left wheels	kg·m^2
25	*YY* product of inertia of rear left wheels	kg·m^2
26	*ZZ* product of inertia of rear left wheels	kg·m^2
27	*XX* product of inertia of rear right wheels	kg·m^2
28	*YY* product of inertia of rear right wheels	kg·m^2
29	*ZZ* product of inertia of rear right wheels	kg·m^2
30	*X*-coordinate of front left wheel axis	m
31	*Y*-coordinate of front left wheel axis	m
32	*Z*-coordinate of front left wheel axis ($u_{z,\text{wheel}}$)	m
33	*X*-coordinate of front right wheel axis	m
34	*Y*-coordinate of front right wheel axis	m
35	*Z*-coordinate of front right wheel axis	m

Table 5.5: Design parameters of the coach sensitivity analysis.

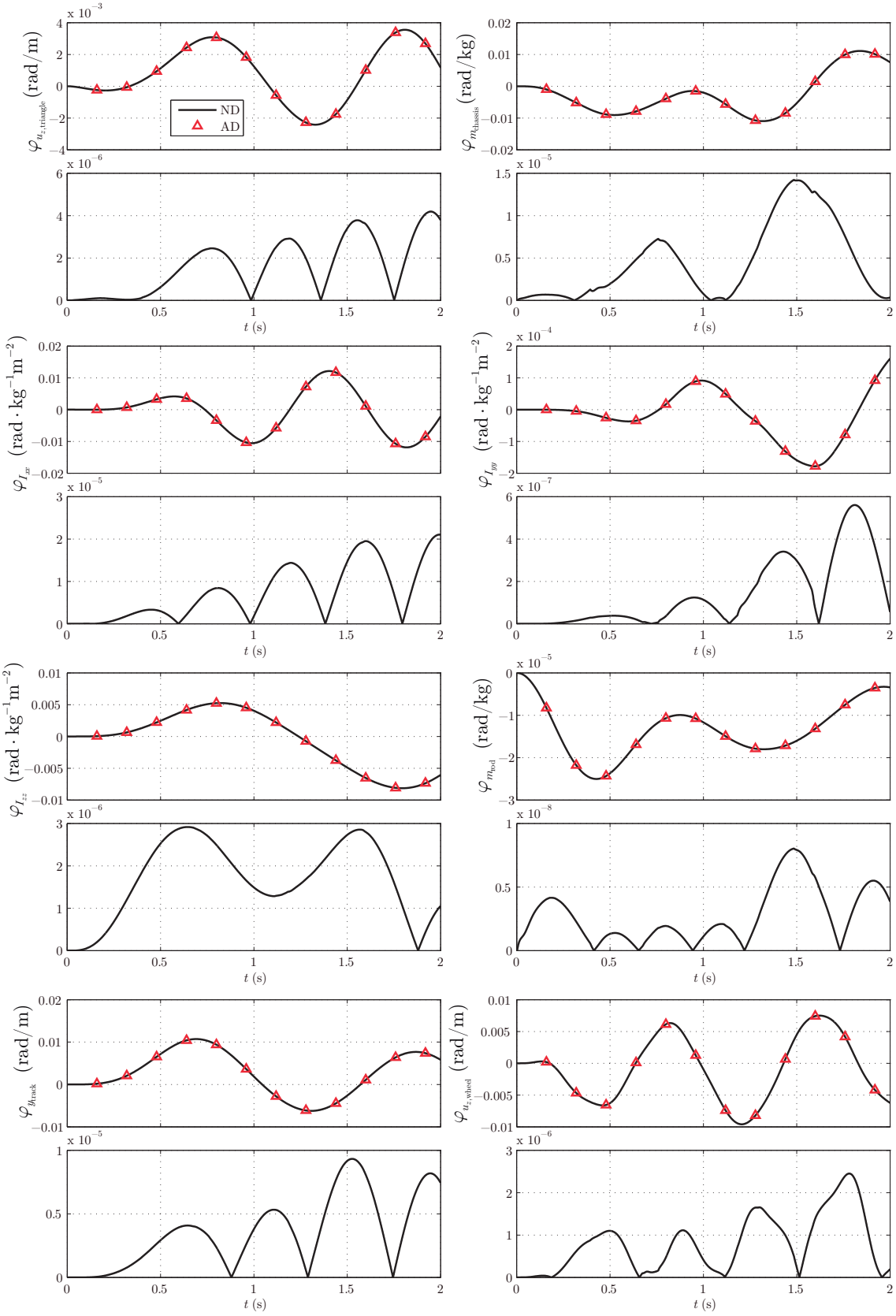


Figure 5.7: Roll sensitivities in the coach maneuver.

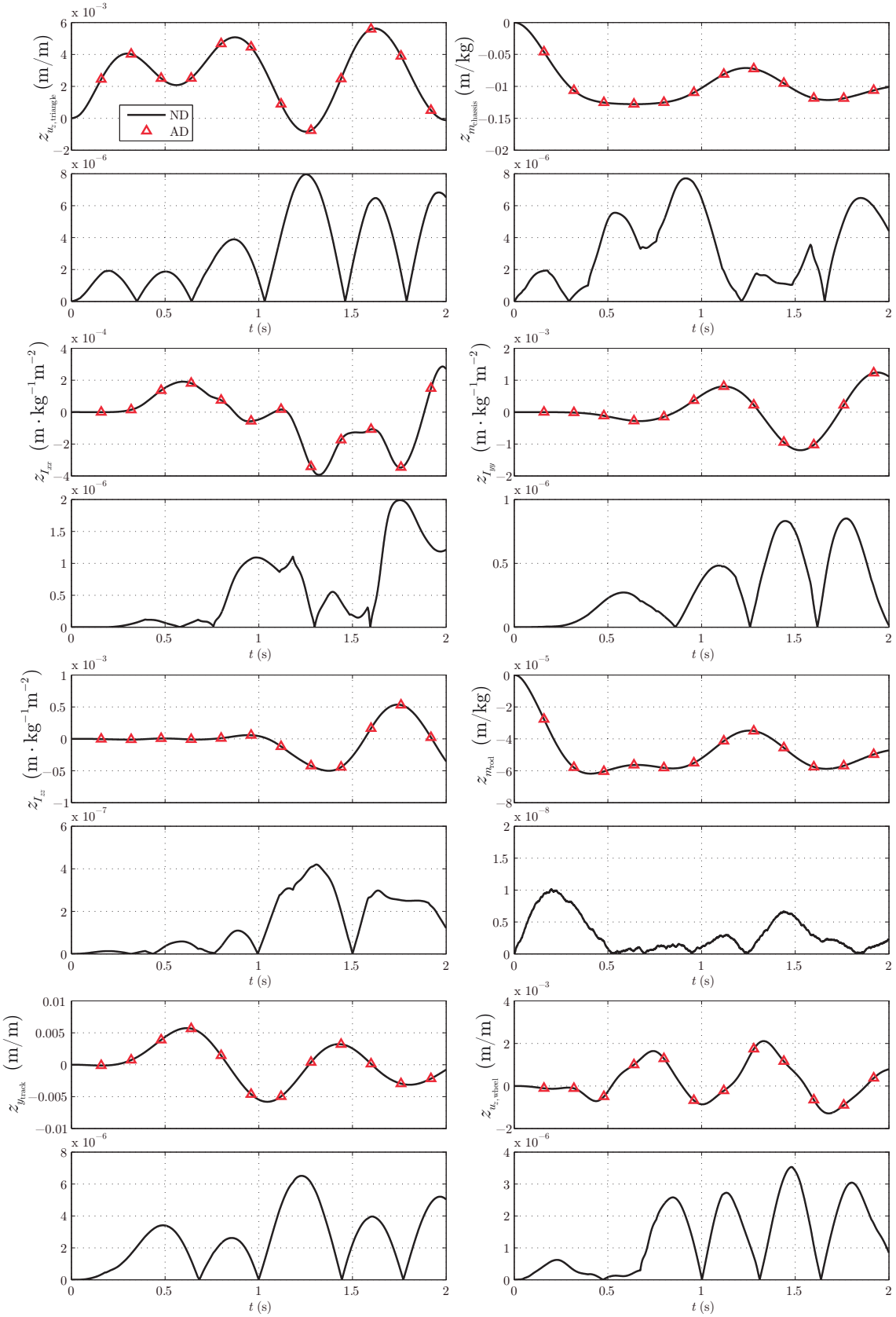


Figure 5.8: *Z*-coordinate sensitivities in the coach maneuver.

One of the most interesting features of an accurate sensitivity analysis is the ability to perform relevance analyses for large sets of parameters, thus solving this problem in a systematic yet automatic manner. In the coach case, a relevance analysis has been performed for two different maneuvers (the previously presented turn maneuver and a straight maneuver), so that the differences can be easily seen. The sensitivity analysis returns a huge amount of state sensitivities, namely, the derivatives of positions, velocities and accelerations w.r.t. all parameters, over time. As way of example, the relevance of parameter b is hereby defined as the maximum absolute value of its position state sensitivities:

$$\text{relevance} = \max\left(\left|\mathbf{z}_b(t)\right|\right) \tag{5.45}$$

Table 5.6 shows the design parameters sorted by their relevance, from the most relevant to the least relevant. Next to the parameter ID, the value of the relevance and the corresponding position coordinate are shown. To identify the latter, the reader may check the coordinate names in Figure 3.20. The list shows that parameters #20 and #21 have virtually no influence in neither the turn maneuver nor the straight maneuver, and thus could be removed from the parameter set. It also shows that, in this case, even though the ordering of parameters changes, the most relevant parameters in both maneuvers are the same. Obviously, the specific system inputs (in this case, the specific maneuver) change the relevance of the design parameters. Finally, note that, due to the small value of the sensitivities, only accurate differentiation methods like AD would be amenable to such kind of relevance analysis.

Elapsed times of the full sensitivity analysis with 35 parameters are presented in Table 5.7. The effect of the number of parameters on the computation time is shown in Figure 5.9(c), where the sensitivity analysis has been run with 1, 18 and 35 parameters. Although new AD techniques are being investigated, current ND computation times are shorter than AD times in this specific example. On the other hand, ND errors are again larger than AD errors (although acceptable, depending on the specific application), but most importantly, they are difficult to estimate and control.

5.6.4 Discussion

The ND approach and the DDM for the solution of global sensitivities have been explored and evaluated. The ND approach loops around the forward dynamics simulation, and its numerical cost is proportional to the number of design parameters. The constant of proportionality depends on the chosen differentiation formula. In the case of central differences, this approach requires $2 \times n_{\text{dp}}$ executions of the forward dynamics. Even if the optimal perturbation size is used, considerable (and unavoidable) numerical errors are expected.

Turn maneuver			**Straight line**		
#	Value	Coordinate	#	Value	Coordinate
31	$2.82\times10^{+1}$	F_L_WHEEL	31	$2.81\times10^{+1}$	F_L_WHEEL
34	$2.81\times10^{+1}$	F_R_WHEEL	34	$2.81\times10^{+1}$	F_R_WHEEL
2	$2.53\times10^{+0}$	R_L_WHEEL_2	2	$2.40\times10^{+0}$	R_R_WHEEL_2
11	$1.01\times10^{+0}$	R_R_WHEEL_2	11	$1.04\times10^{+0}$	R_R_WHEEL_2
22	8.02×10^{-1}	F_R_WHEEL	22	7.79×10^{-1}	F_L_WHEEL
23	7.38×10^{-1}	F_R_WHEEL	23	7.79×10^{-1}	F_R_WHEEL
35	4.66×10^{-1}	CH_AUX2_DISP_Y	35	4.26×10^{-1}	CH_AUX2_DISP_Y
32	4.47×10^{-1}	CH_AUX2_DISP_Y	32	4.26×10^{-1}	CH_AUX2_DISP_Y
10	3.64×10^{-1}	F_R_WHEEL	10	3.40×10^{-1}	F_R_WHEEL
6	3.13×10^{-1}	R_L_WHEEL_2	6	3.00×10^{-1}	R_R_WHEEL_2
5	2.72×10^{-1}	RS_AUX2	15	8.67×10^{-2}	L_S_TRIANG
16	1.48×10^{-1}	F_L_WHEEL	17	8.66×10^{-2}	R_S_TRIANG
1	1.31×10^{-1}	CH_AUX2_DISP_Y	33	5.73×10^{-2}	CH_AUX2_DISP_Y
17	1.12×10^{-1}	RST_AUX1	30	5.73×10^{-2}	CH_AUX2_DISP_Y
15	8.79×10^{-2}	LST_AUX1	1	4.64×10^{-2}	L_S_TRIANG
12	8.31×10^{-2}	R_L_WHEEL_2	19	4.64×10^{-2}	R_S_TRIANG
30	8.01×10^{-2}	CH_AUX2_DISP_Y	14	2.62×10^{-3}	R_R_WHEEL_2
19	7.31×10^{-2}	F_R_WHEEL	13	2.62×10^{-3}	R_L_WHEEL_2
18	7.02×10^{-2}	F_R_WHEEL	18	1.95×10^{-3}	R_S_TRIANG
33	6.04×10^{-2}	F_R_WHEEL	16	1.94×10^{-3}	L_S_TRIANG
3	4.75×10^{-2}	R_R_WHEEL_2	12	1.79×10^{-3}	F_R_WHEEL
4	6.25×10^{-3}	F_R_WHEEL	28	2.18×10^{-5}	R_R_WHEEL
13	2.66×10^{-3}	R_L_WHEEL_2	25	2.17×10^{-5}	R_L_WHEEL
14	2.51×10^{-3}	R_R_WHEEL_2	4	6.52×10^{-6}	F_R_WHEEL
9	6.53×10^{-4}	RS_AUX2	3	5.65×10^{-6}	F_R_WHEEL
25	3.97×10^{-4}	R_L_WHEEL_2	5	2.76×10^{-6}	CH_AUX2_DISP_Y
28	3.78×10^{-4}	R_R_WHEEL_2	7	3.55×10^{-8}	R_R_WHEEL_2
7	1.64×10^{-4}	R_L_WHEEL_2	8	2.49×10^{-8}	R_L_WHEEL
8	1.82×10^{-5}	F_R_WHEEL	24	1.42×10^{-8}	R_L_WHEEL_2
27	1.12×10^{-5}	R_R_STAB	9	1.07×10^{-8}	F_R_WHEEL
29	1.09×10^{-5}	R_R_STAB	27	1.07×10^{-8}	R_R_WHEEL
26	1.03×10^{-5}	R_R_STAB	29	1.07×10^{-8}	R_L_WHEEL_2
24	1.02×10^{-5}	R_R_STAB	26	7.11×10^{-9}	R_L_WHEEL_2
20	$0.00\times10^{+0}$	R_R_STAB	20	$0.00\times10^{+0}$	R_R_STAB
21	$0.00\times10^{+0}$	R_R_STAB	21	$0.00\times10^{+0}$	R_R_STAB

Table 5.6: Analysis of parameter relevance.

Elapsed time (s)	35 parameters	
	ND	AD
Dynamics	0.694	0.694
Sensitivity	48.112	81.034
Total	48.806	81.728
Sensitivity/Total	99%	99%

Table 5.7: Computation times of the 2-second coach simulation.

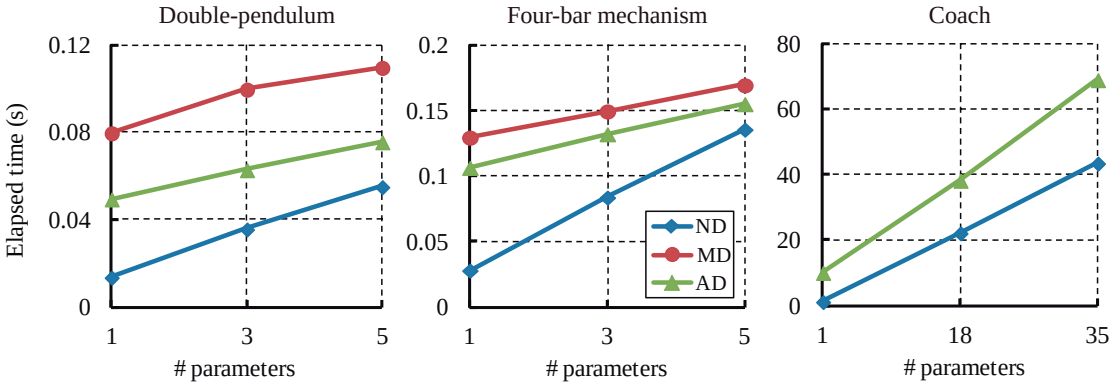


Figure 5.9: Influence of the number of parameters on the computation times.

On the contrary, the AVM returns more accurate results than the ND approach. Even though it was not implemented in this Thesis, its computational cost is proportional to the responses and is independent from the number of design parameters. This can be exploited in cases with a small number of outputs and a large number of design parameters. However, the complexity of implementation and integration are important drawbacks for realistic applications.

The third global approach, namely the DDM, returns accurate results as well, at the cost of solving $(n_{\mathrm{dp}}+1)$ sets of f second-order ODEs sharing the same inertia matrix. In this approach, dynamic and sensitivity equations need to be solved jointly because they are coupled. The derivatives of the terms involved are of medium complexity and need to be calculated by one of three methods: by hand (MD), using symbolic software packages (SD) or using automatic differentiation (AD). Note also that the DDM is very well suited to compute explicit intermediate variables (which are invisible in the adjoint variable method) and to use multiple time point constraints.

The hybrid direct-automatic differentiation approach presented here is based on the state-space computation of independent sensitivities via the DDM and the AD technique. The coordinate partitioning method is used to formulate sensitivity equations. Only independent sensitivities are integrated, and Maggi's strategy is used for both the integration of the motion differential equations and the sensitivity equations. Moreover, as explained in Chapter 2, the use of the path matrix (to formulate and automate the recursivity) and natural coordinates (for

the initial geometry and the constraint enforcement) make the approach even more simple and efficient. With respect to Wang, Haug and Pan, 2005, who also use the coordinate partitioning method to implicitly compute independent sensitivities, in the presented approach no nonlinear system of equations has to be solved. Also, the presented method has all the advantages of explicit time-integration schemes. As far as the numerical experiments are concerned, equally stiff (or may be stiffer) examples are considered here, and the computation times hereby obtained are much shorter.

The solution of three numerical examples (a double-pendulum, a four-bar mechanism and an 18-DOF coach maneuver) have proven the accuracy and efficiency of the presented approach. The sensitivity analysis has been used, in the 35-parameter coach, to perform a parameter relevance analysis, which is expected to help the designer choose an optimal set of parameters. MD and AD yield machine-precision sensitivities, whereas ND, as expected, entails errors difficult to estimate and control. The development time with AD is far shorter than the one with MD, thus being more scalable. However, operator-overloading AD is still less efficient than ND when the number of DOFs is large. Finally, AD appears to have the smallest dependency of the computation time on the number of parameters, at least in the two small examples.

In short, a step has been taken towards a more efficient, robust and simple computation of design sensitivities in the field of multibody dynamics.

Chapter 6

Optimization of the dynamic response

The choice of suspension characteristics within the vehicle design stage requires a trade-off between the components' service life, the handling characteristics and the ride comfort characteristics. Multibody analysis tools play an important role in the design of vehicles, as they help evaluate the dynamics accurately with a short development time and a low computational cost, as opposed to, for instance, dynamic finite element method tools. However, to the author's knowledge, none of the mainstream multibody tools allow for a state-of-the-art optimization of mechanical systems. At most, they include somewhat primitive utilities for the design of experiments based on predefined parameter sets. In cases with many design parameters, these approaches are clearly inefficient.

From a numerical point of view, the dynamic response optimization of mechanical systems is a nonlinear programming problem with nonlinear constraints, and as such, is one of the most complex and challenging optimization problems. It implies dealing with the following aspects:

- Time integration of the mechanical system, including problems like stability, efficiency and accuracy.
- Implementation and (in some cases) coupling of multibody simulation software and optimization algorithms.
- Solution of the sensitivity equations for gradient-based methods, or error assessment if only finite differences are employed.
- Characterization of the dynamic response, including the definition of the system inputs, the case studies and the quality metrics.
- Multi-objective optimization of conflicting objectives.

Some of these issues have already been addressed in the previous chapters. The ones related to mathematical optimization are tackled now. Obviously, a com-

plete analysis of the wide range of optimization methods and techniques applied to mechanical design is out of the scope of this Thesis. Instead, an outline of the most representative tools is presented, and the design optimization of the coach presented in Chapter 3 is carried out. This chapter brings together ideas from the five previous chapters, and as such, it constitutes the height of this Thesis.

6.1 Optimization methods

The available optimization methods for the solution of mechanical engineering problems are fairly numerous. Any method that is able to solve nonlinear optimization problems with constraints should be able to deal with the present case. Furthermore, the particular implementation developed in this Thesis (based on the C/C++ programming of core functions through MATLAB's MEX-functions) allows the interfacing of the objective functions and the constraint equations with any program supporting MATLAB and C/C++, which broadens the range of candidate optimization libraries.

In this chapter, the emphasis has been set on optimization algorithms that have been intensively tested in the literature and have proven to be effective. Specifically, MATLAB's `Optimization` and `Global Optimization` toolboxes have been used, because they are written by specialists and they very closely represent the state of the art on optimization methods. In order to cover, to some extent, the most important families of optimization algorithms, three different types have been considered: *local*, *global* and *multi-objective* methods. For additional theoretical details on these methods and many others, the reader might want to see Fletcher, 1987, Arora, 1989, Kelley, 1999, Venkataraman, 2002, Mastinu, Gobbi and Miano, 2006, and Nocedal and Wright, 2006.

6.1.1 Local methods

Local optimization approaches rely on the first- and higher-order derivative information of the starting and successive points to sequentially improve the objective function. Since only the local information is used on each iteration, the quality of the initial set of design parameters has a great influence on the outcome of the simulation. Also, local methods usually converge to the function minimum closest to the starting point, which means general non-convex problems with multiple local minima could yield non-optimal solutions. This section presents some of the most relevant local optimization methods, that will eventually be applied to the dynamic response optimization problem at hand.

Based on the concepts and notation from Section 1.3.2 and Chapter 5, a generic nonlinear optimization problem with inequality and equality constraint equations can be written as:

$$\begin{aligned} \min_{\mathbf{b}} \ & \Psi_0 = f(\mathbf{b}) \\ \text{s.t.} \ & \boldsymbol{\Psi}^{\text{in}}(\mathbf{b}) \leq \mathbf{0} \\ & \boldsymbol{\Psi}^{\text{eq}}(\mathbf{b}) = \mathbf{0} \end{aligned} \tag{6.1}$$

where $\boldsymbol{\Psi}^{\text{in}} \in \mathbb{R}^{m_{\text{in}}}$ are the inequality constraints and $\boldsymbol{\Psi}^{\text{eq}} \in \mathbb{R}^{m_{\text{eq}}}$ are the equality constraints. The very well-known Karush-Kuhn-Tucker necessary conditions for a local solution point $\mathbf{b}^*$ are the following:

$$\boldsymbol{\nabla} f(\mathbf{b}^*) + \boldsymbol{\nabla}\boldsymbol{\Psi}^{\text{in}}(\mathbf{b}^*)^T \boldsymbol{\mu} + \boldsymbol{\nabla}\boldsymbol{\Psi}^{\text{eq}}(\mathbf{b}^*)^T \boldsymbol{\lambda} = \mathbf{0} \tag{6.2}$$

$$\boldsymbol{\mu}^T \boldsymbol{\Psi}^{\text{in}}(\mathbf{b}^*) = 0 \tag{6.3}$$

$$\boldsymbol{\mu} \geq \mathbf{0} \tag{6.4}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{m_{\text{in}}}$ and $\boldsymbol{\lambda} \in \mathbb{R}^{m_{\text{eq}}}$ are multiplier vectors.

Penalty and barrier methods One of the most direct approaches for the solution of constrained nonlinear programming problems consists of including a penalty term in the objective function, making it grow rapidly with the violation of the constraints. Let us pose the following optimization problem:

$$\begin{aligned} \min_{\mathbf{b}} \ & \Psi_0 = f(\mathbf{b}) \\ \text{s.t.} \ & \boldsymbol{\Psi}^{\text{in}}(\mathbf{b}) \leq \mathbf{0} \end{aligned} \tag{6.5}$$

where $\boldsymbol{\Psi}^{\text{in}}(\mathbf{b}) \in \mathbb{R}^m$ is a set of m inequality constraints. A *penalty function* can be defined as any continuous function $\phi(\mathbf{b})$ satisfying $\phi(\mathbf{b}) \geq 0$ for all $\mathbf{b} \in \mathbb{R}^n$ and $\phi(\mathbf{b}) = 0$ when $\mathbf{b}$ is feasible, for instance:

$$\phi(\mathbf{b}) = \sum_{i=1}^{m} \left\{ \max\left[0,\ \Psi_i^{\text{in}}(\mathbf{b})\right] \right\}^2 \tag{6.6}$$

The optimization problem in Eq. (6.5) can then be formulated as an unconstrained optimization problem in the form:

$$\min_{\mathbf{b}} \ \Psi_0 = f(\mathbf{b}) + \alpha\, \phi(\mathbf{b}) \tag{6.7}$$

where α is a large, user-defined penalty value. The choice of α depends on the particular problem under study. A very large value could cause bad conditioning, and a small value would not enforce the constraints sufficiently. This setback can be overcome by sequentially solving the problem with growing values of the penalty value. Other forms of penalty functions can be defined, as well as penalty functions for equality constraints.

A similar approach is followed by *barrier methods*, except they enforce constraints more strictly, not allowing points to be outside the feasible set, thus only considering *interior points*. Barrier functions penalize the solutions as they approach the boundaries, and they do not allow them to be trespassed. Penalty and barrier methods are, in general, easy to implement but not very efficient.

Sequential quadratic programming It is sometimes referred to as *Lagrange-Newton method* because it applies the Newton-Raphson method to find a stationary point of the Lagrange function. It was developed by Wilson, Han and Powell in the sixties, and it is one of the most popular methods for the solution of nonlinear optimization problems with constraints. In each iteration of the SQP method, a quadratic optimization method is solved.

First, let us consider the optimization of a quadratic problem with linear equality constraints, which can be written as:

$$\begin{aligned} \min_{\mathbf{b}} \quad & \tfrac{1}{2}\mathbf{b}^T\mathbf{H}\mathbf{b} + \mathbf{g}^T\mathbf{b} \\ \text{s.t.} \quad & \mathbf{c}(\mathbf{b}) \equiv \mathbf{A}^T\mathbf{b} = \mathbf{h} \end{aligned} \tag{6.8}$$

where $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{H} \in \mathbb{R}^{n\times n}$, $\mathbf{g} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{n\times m}$ and $\mathbf{h} \in \mathbb{R}^m$. The associated Lagrange function would be:

$$L(\mathbf{b},\boldsymbol{\lambda}) = f(\mathbf{b}) - \boldsymbol{\lambda}^T\mathbf{c}(\mathbf{b}) = \frac{1}{2}\mathbf{b}^T\mathbf{H}\mathbf{b} + \mathbf{g}^T\mathbf{b} - \boldsymbol{\lambda}^T(\mathbf{A}^T\mathbf{b} - \mathbf{h}) \tag{6.9}$$

Matrix $\mathbf{H}$ is assumed to be positive definite. Thus, the problem is strictly convex, and the global minimum will be a stationary point of function L. Applying the Karush-Kuhn-Tucker theorem by differentiating with respect to the design vector, $\mathbf{b}$, and the Lagrange multipliers, $\boldsymbol{\lambda}$, at the local minimizer, $\mathbf{b}^*$:

$$L_{\mathbf{b}}(\mathbf{b}^*,\boldsymbol{\lambda}^*) = \mathbf{H}\mathbf{b}^* + \mathbf{g} - \mathbf{A}\boldsymbol{\lambda}^* = \mathbf{0} \tag{6.10}$$

$$L_{\boldsymbol{\lambda}}(\mathbf{b}^*,\boldsymbol{\lambda}^*) = \mathbf{A}^T\mathbf{b}^* - \mathbf{h} = \mathbf{0} \tag{6.11}$$

In matrix form, Eqs. (6.10)–(6.11) can be expressed as:

$$\begin{bmatrix} \mathbf{H} & -\mathbf{A} \\ -\mathbf{A}^T & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{b} \\ \boldsymbol{\lambda} \end{Bmatrix} = -\begin{Bmatrix} \mathbf{g} \\ \mathbf{h} \end{Bmatrix} \tag{6.12}$$

This $(n+m)\times(n+m)$ system of linear equations can be solved directly. We can now introduce inequality constraints together with the concept of *active constraints*. An active constraint is an inequality constraint whose value is zero, and $\Gamma(\mathbf{b})$ are the s indices of the active constraints. The *active set method* adds active constraints to the set of equality constraints, ignores the inactive inequality constraints and solves an augmented $(n+m+s)\times(n+m+s)$ version of Eq. (6.8) for a temporary solution $(\mathbf{b}_{\text{eq}},\boldsymbol{\lambda}_{\text{eq}})$. As $(\mathbf{b}_{\text{eq}},\boldsymbol{\lambda}_{\text{eq}})$ might violate the ignored constraints, the feasible solution which is closest to $(\mathbf{b}_{\text{eq}},\boldsymbol{\lambda}_{\text{eq}})$ is selected.

Now let us approach the nonlinear problem from another angle. Consider a generic optimization problem with objective function $f(\mathbf{b})$ and equality constraint equations $\mathbf{c}(\mathbf{b})$:

$$\begin{aligned} \min_{\mathbf{b}} \quad & f(\mathbf{b}) \\ \text{s.t.} \quad & \mathbf{c}(\mathbf{b}) = \mathbf{0} \end{aligned} \tag{6.13}$$

The corresponding Lagrange function can be linearized around point $(\mathbf{b}, \boldsymbol{\lambda})$ as:

$$L(\tilde{\mathbf{b}}, \tilde{\boldsymbol{\lambda}}) \equiv L(\mathbf{b} + \boldsymbol{\delta}^b, \boldsymbol{\lambda} + \boldsymbol{\delta}^\lambda) \approx$$

$$\approx L(\mathbf{b}, \boldsymbol{\lambda}) + \begin{Bmatrix} \boldsymbol{\delta}^b & \boldsymbol{\delta}^\lambda \end{Bmatrix} \begin{Bmatrix} L_{\mathrm{b}} \\ L_{\boldsymbol{\lambda}} \end{Bmatrix} + \frac{1}{2} \begin{Bmatrix} \boldsymbol{\delta}^b & \boldsymbol{\delta}^\lambda \end{Bmatrix} \begin{bmatrix} L_{\mathrm{bb}} & L_{\boldsymbol{\lambda}\mathrm{b}} \\ L_{\mathrm{b}\boldsymbol{\lambda}} & L_{\boldsymbol{\lambda}\boldsymbol{\lambda}} \end{bmatrix} \begin{Bmatrix} \boldsymbol{\delta}^b \\ \boldsymbol{\delta}^\lambda \end{Bmatrix} \tag{6.14}$$

Applying the necessary stationary conditions (6.10)–(6.11), the approximation in Eq. (6.14), and $\boldsymbol{\delta}^\lambda = \tilde{\boldsymbol{\lambda}} - \boldsymbol{\lambda}$, the following linear system holds:

$$\begin{bmatrix} L_{\mathrm{bb}} & -\mathbf{c}_{\mathrm{b}}^T \\ -\mathbf{c}_{\mathrm{b}} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \boldsymbol{\delta}^b \\ \tilde{\boldsymbol{\lambda}} \end{Bmatrix} = - \begin{Bmatrix} f_{\mathrm{b}} \\ -\mathbf{c} \end{Bmatrix} \tag{6.15}$$

$$\tilde{\mathbf{b}} = \mathbf{b} + \boldsymbol{\delta}^b \tag{6.16}$$

When solved, this system provides a new optimum approximation $(\tilde{\mathbf{b}}, \tilde{\boldsymbol{\lambda}})$. At this point, one may notice that systems (6.15) and (6.12) look very similar. Comparing terms, Eq. (6.15) can be seen as the solution to the following quadratic problem:

$$\begin{aligned} \min_{\boldsymbol{\delta}^b} \quad & \tfrac{1}{2} \boldsymbol{\delta}^{bT} L_{\mathrm{bb}} \boldsymbol{\delta}^b + f_{\mathrm{b}}^T \boldsymbol{\delta}^b + f(\mathbf{b}) \\ \text{s.t.} \quad & \mathbf{c}_{\mathrm{b}}^T \boldsymbol{\delta}^b + \mathbf{c}(\mathbf{b}) = \mathbf{0} \end{aligned} \tag{6.17}$$

where $f(\mathbf{b})$ has been added to the objective function and does not alter the problem. Equation (6.17)a constitutes a second-order approximation to $f(\mathbf{b})$, and Eq. (6.17)b is a first-order approximation to $\mathbf{c}(\mathbf{b})$. Globally, Eq. (6.17) is a quadratic optimization approximation to Eq. (6.13). When inequality constraints are present, Eq. (6.17) can be augmented with the active set of inequality constraints.

The convergence of the SQP method is quadratic when exact derivatives are used. Also, the computation of second-order derivatives L_{bb} and the solution of linear systems can be eliminated, for instance, using a BFGS approach (Nocedal and Wright, 2006).

6.1.2 Global methods

The second large family of optimization methods are global algorithms. Among them, stochastic methods make up an important portion. Their main advantage over local methods is that they can span the whole design space, thus increasing the chances of finding the global minimum of the function. However, they do so at the cost of numerous objective function evaluations, meaning they can turn out to be very inefficient.

Genetic algorithm (GA) It is one of the most important methods within *evolutionary computation*. It was first used by Holland, 1962, but was not applied to optimization problems until the late eighties. The idea of this method is to apply the principles of biological evolution to the improvement of the solu-

tion candidates. Basically, the fittest candidates are selected on each iteration of the process. In the biological processes, the chromosomes (design parameters) mutate randomly and recombine to generate a better gene structure (objective function). GA methods have been used in all sorts of scientific applications, among which discrete programming applications take an important role.

GA algorithms are stochastic methods, meaning that some sort of randomness is involved, thus they are suited to find global solutions. GAs are most useful in ill-conditioned, discontinuous, non-differentiable and discrete problems. They are based on the generation of successive populations of candidates and, unlike other global methods, each candidate is totally unrelated to the others, meaning that there is no step size calculation and therefore a specific solution cannot be improved around its neighborhood.

Chromosomes are design parameter vectors, and can be handled directly or with some kind of mapping. The *fitness function* is used to evaluate and quantify the quality of the population, and is obviously related to the objective function. There are several criteria for the selection of the next generation. An initial population (or set of parameter vectors) can usually be defined. *Genetic operators* are then used to define the new population from the current one, of which *crossover* (or *recombination*) and *mutation* are the most common ones. In the first case, characteristics from both parents are blended into the new candidate, and in the second, a random value is introduced into a random element of the design vector. Finally, random vectors (*immigrants*) are introduced into the population, so that the search for a global minimum goes on.

Simulated annealing (SA) It was first applied to optimization by Kirkpatrick, Gelatt and Vecchi, 1983, as a heuristic extension of the simulated annealing principle. This principle is based on the cooling of metals, and on how the thermal equilibrium of a set of atoms is reached after being heated.

The algorithm starts by defining a random search direction and a predefined step size, α. If the objective function is reduced, then $p = 1$; otherwise, $p = e^{-\beta \Delta f}$. The step will be accepted only if he value of p is greater than a random number $r \in [0,1)$. Two conclusions can be easily drawn: first, the SA method is fairly straightforward to implement; second, the values of α and β are very important for the search of the global solution. The value of β is called *annealing temperature*, and it will in fact determine the probability that a worse solution is accepted.

Further improvements can be implemented in the SA method. Among others, the *annealing schedule* allows the value of β to change over time, e.g. to relax the annealing in the first stages and harden it as the optimization goes forward. Due to its stochastic nature, the solution does not improve iteratively, and the number of iterations required cannot be estimated beforehand. Finally, note

that when constraints are present, the SA method is expected to take a significantly larger number of iterations to converge.

Direct search methods Similar to stochastic methods, direct-search methods do not need gradient information. They are based on a set of objective function evaluations, which they use to continue the sampling. Some examples of direct-search algorithms are the Nelder-Mead simplex algorithm, the Hooke-Jeeves algorithm and the multi-directional search algorithm.

6.1.3 Multi-objective methods

It is somewhat common in engineering design problems to have several conflicting objectives. In them, the designer plays a more important role than in standard nonlinear programming problems, mainly because multi-objective optimization problems do not have a unique solution. Very often the objectives are weighted at the designer's discretion and added into a single objective function, which can then be optimized using standard methods.

In the vehicle dynamics case under consideration, it would be desirable to tackle both handling and comfort behaviors simultaneously in the optimization process, so that neither one is favored. In general, the methods for the solution of such multi-objective problems are involved, time-consuming, and always assume some sort of discretionary weighting of the objective functions. Here, one of the most general approaches has been followed, namely the one based on the concept of Pareto optimality (see Mastinu, Gobbi and Miano, 2006).

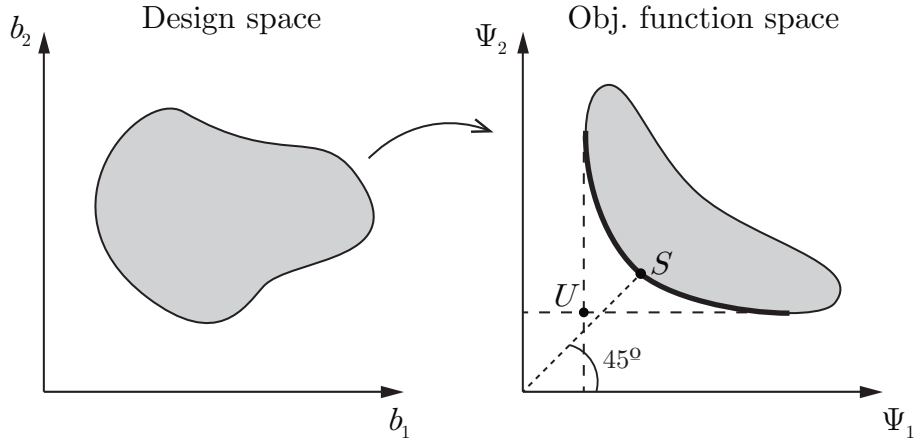


Figure 6.1: Design and objective function spaces.

The *Pareto-optimal set* is defined as the set of solution points at which the improvement of one objective function involves the worsening of at least one of the others. This concept is widely used in engineering and economics. Figure 6.1 shows the Pareto front as a thick line in the objective function space, in a two-parameter, two-objective case. Points not belonging to the Pareto-front accept an improvement (a minimization) of both objective functions simultaneously.

All Pareto-optimal points are valid solutions to the optimization problem, which means multi-objective problems have infinite solutions. Point U represents the *utopia point*, which would be one where all objective functions held their best value simultaneously. Pareto-optimal optimization methods deal with efficient ways of finding the Pareto front and/or choosing the a specific solution within.

There are several strategies to find the Pareto-optimal set and/or a specific Pareto-optimal solution. The *exhaustive* approach would consist of evaluating the objective functions for a set of equally-spaced points spanning the entire design space. The computational cost of this approach prevents its use on real-life optimization problems. More refined approaches would consist of defining less dense sequences of design points distributed along the design space.

The second large group of methods consists of applying local optimization methods to a weighted sum of objective functions. If none of the objective functions are to be favored (or the information about the importance of each objective is not available), the intersection of the first quadrant bisector with the Pareto front can be chosen as the objective point (point S in Figure 6.1). Mathematically, this problem would be equivalent to the following ones:

$$\min_{\mathbf{b}} \max_{i} \Psi_i(\mathbf{b}), \quad i = 1,...,n_{\text{of}} \tag{6.18}$$

and

$$\begin{aligned} &\min \gamma \\ &\text{s.t.} \begin{cases} \Psi_1 \leq \gamma \\ \Psi_2 \leq \gamma \\ \dots \\ \Psi_{n_{\text{of}}} \leq \gamma \end{cases} \end{aligned} \tag{6.19}$$

where Ψ_i are the objective functions. See Kanarachos, 2012, for a somewhat similar application to vehicle dynamics.

The last family of methods for the solution of multi-objective problems are Pareto GAs, which use global optimization methods to compute the Pareto front and/or a solution on it. In this Thesis, these methods have not been used because MATLAB's Pareto GAs could not handle nonlinear optimization constraints, which, as will be proven, are essential for the dynamic response optimization of real-life vehicles.

Finally, similarly to the relevance analysis presented in Section 5.6.3, multi-objective problems can benefit from sensitivity analyses. Specifically, global sensitivity analyses can be used to reduce the number of objective functions by entering large variations of the design parameters and detecting strongly correlated objective functions. Also, sensitivity analyses in the Pareto front can help the

designer understand the relationship between objective functions in the neighborhood of the Pareto-optimal solutions.

6.2 Parameter identification

The first dynamic response optimization problem solved here is a rather simple parameter identification problem. Even though the identification of system parameters, as explained, constitutes a whole different field of multibody system analysis, one of the typical approaches to solve these problems is based on mathematical optimization (for instance, Serban and Freeman, 2001). Some ideas presented here will also be applied in the following optimization problems.

6.2.1 Problem definition

Models of real-life mechanical systems often have parameters difficult to measure, calculate or estimate. Even if they can be measured, the process might involve disassembling, which is always costly and laborious. In the specific field of vehicle dynamics, for example, it might not be reasonable to calculate parameters like the bodywork torsion stiffness or the bodywork inertia tensor. When these types of parameters are key for the dynamic response, a systematic identification becomes highly desirable.

Objective function Let $\mathbf{b}_{\text{ini}}$ be a set of initial (estimated) parameters and $\mathbf{b}^*$ the vector of identified parameters. By using the response data from a specific real maneuver, it is possible to vary the model parameters in a way that the difference between the real test and the simulated maneuver is minimized. The solution of such a problem would be the vector of identified parameters $\mathbf{b}^*$. From an optimization point of view, the objective function can be written as:

$$\min_{\mathbf{b}} \ \Psi_0 = \\ = \sqrt{\int_{t_i}^{t_f} \left[(\hat{x}-x)^2 + (\hat{y}-y)^2 + (\hat{z}-z)^2 + (\hat{\varphi}-\varphi)^2 + (\hat{\theta}-\theta)^2 + (\hat{\psi}-\psi)^2 \right] dt} \tag{6.20}$$

where the variable dependencies have been omitted for the sake of clarity. All translational and rotational bodywork position variables, x, y, z, φ, θ and ψ, depend on t, $\mathbf{z}$, $\dot{\mathbf{z}}$, $\ddot{\mathbf{z}}$ and $\mathbf{b}$. Positions $\hat{x}$, $\hat{y}$, $\hat{z}$, $\hat{\varphi}$, $\hat{\theta}$ and $\hat{\psi}$ correspond to the test (reference) data, and only depend on t. Equation (6.20) is nothing but the RMS value of the difference between the position responses of the model and the real test. Other responses like contact forces or accelerations could have been selected as well. The computation of the objective function gradient is performed using the hybrid direct-automatic differentiation approach, as thoroughly described in the previous chapters. By differentiating the objective function with respect to the parameters, the following expression can be obtained:

$$2\Psi_0 \frac{d\Psi_0}{d\mathbf{b}} = -\int_{t_i}^{t_f} \Big[2(\hat{x}-x)\frac{dx}{d\mathbf{b}} + 2(\hat{y}-y)\frac{dy}{d\mathbf{b}} + 2(\hat{z}-z)\frac{dz}{d\mathbf{b}} + \\ + 2(\hat{\varphi}-\varphi)\frac{d\varphi}{d\mathbf{b}} + 2(\hat{\theta}-\theta)\frac{d\theta}{d\mathbf{b}} + 2(\hat{\psi}-\psi)\frac{d\psi}{d\mathbf{b}} \Big] dt \tag{6.21}$$

which, once the sensitivity analysis has been carried out, can be easily solved for the objective function gradient in terms of the state sensitivities.

Optimization constraints A meaningful mathematical optimization requires the definition of constraint equations, especially in the case of real-life systems like the present one. The designer must detect unrealistic solutions and prevent them by introducing variable boundaries and simulation constraints. Within the field of vehicle response optimization, the literature proposes a great variety of constraint types. To name a few:

- Load or deflection of tires: Besselink and Van Asperen, 1994, Baumal, McPhee and Calamai, 1998.
- Tire hop: Thoresson, 2007.
- Elongation of springs: Besselink and Van Asperen, 1994.
- Displacement of a point: Baumal, McPhee and Calamai, 1998, Gonçalves, 2002, Naudé and Snyman, 2003, Gonçalves and Ambrósio, 2005.
- Acceleration of a point: Besselink and Van Asperen, 1994, Gonçalves, 2002.
- Roll angle: Andersson and Eriksson, 2004.

The most important constraint within general vehicle maneuvers is to enforce grip during the dynamic maneuver. During the optimization process, a random change of the suspension parameters can cause an excessive lateral and/or longitudinal tire slip, which means that the vehicle is going to lose stability. This situation must be prevented by all means. Here, one grip constraint is introduced for each tire. Grip constraints are based on Pacejka's model of the side force (see Section 3.1.5). For a growing side slip angle, the side force grows up to a maximum, after which the side force starts decreasing and the tire loses lateral stability. Grip constraints are formulated so that, in all six tires, the slope of the side force, F_y, w.r.t. the side slip angle, α, is always positive:

$$\mathbf{\Psi}^{\text{grip}} \equiv -\begin{Bmatrix} \max\limits_t \dfrac{\partial F_{y,1}}{\partial \alpha}(\alpha,t) \\ \max\limits_t \dfrac{\partial F_{y,2}}{\partial \alpha}(\alpha,t) \\ \cdots \\ \max\limits_t \dfrac{\partial F_{y,6}}{\partial \alpha}(\alpha,t) \end{Bmatrix} \leq \mathbf{0} \tag{6.22}$$

where F_y is the side tire force.

In the initial configuration, the side force slopes during the DLC maneuver are fully compliant with the grip condition, as can be seen in Figure 6.2. This figure shows the slope over time of each of the six coach tires.

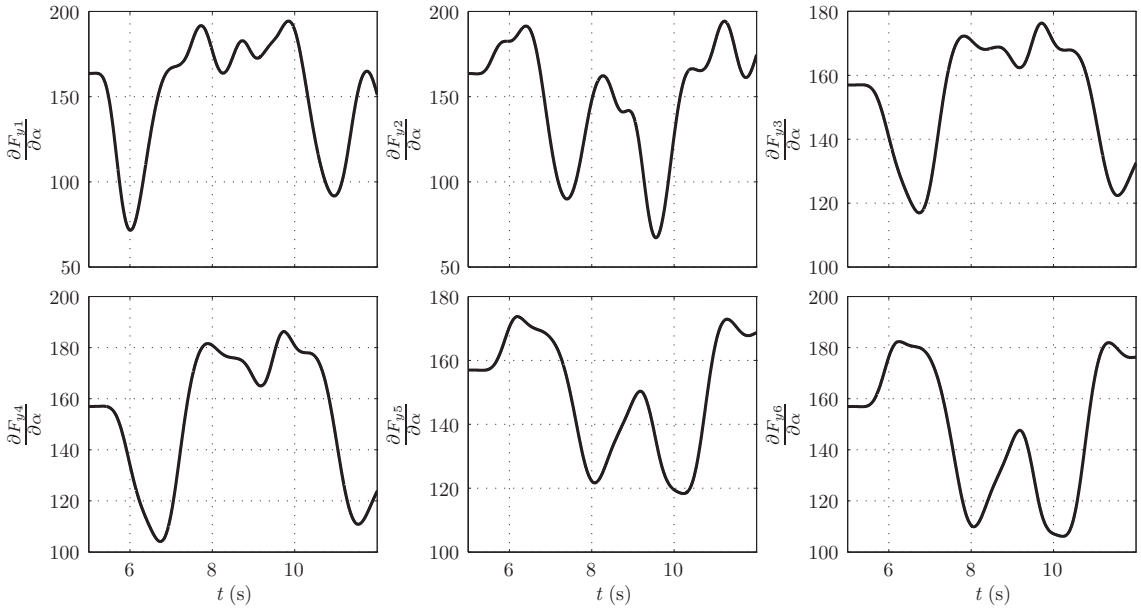


Figure 6.2: Side force slopes during the DLC maneuver.

Another basic requirement of the suspension in this case is that the wheels do not lift from the ground. Safety in the case of coaches and buses is of paramount importance, and since only safe suspension setups are analyzed in this Thesis, safety limits are neither reached nor sought. However, other simulations like racing car tests and simulations with more aggressive scenarios would require accounting for the tire lifting. A possible formulation of the wheel contact condition is to enforce positive values of the six tire normal forces:

$$\mathbf{\Psi}^{\text{hop}} = -\begin{Bmatrix} \max\limits_{t} F_{z,1}(t) \\ \max\limits_{t} F_{z,2}(t) \\ \cdots \\ \max\limits_{t} F_{z,6}(t) \end{Bmatrix} \leq \mathbf{0} \tag{6.23}$$

where F_z is the normal contact force. Finally, box constraints are imposed on the design variables, in order to prevent unrealistic parameter values. Also, according to the upper and lower bounds, the parameters are normalized to improve the conditioning of the numerical problem:

$$b_i = \frac{b_c - b_l}{b_u - b_l} \tag{6.24}$$

where b_c is the current value of the parameter, b_l is the lower bound and b_u is the upper bound. Note that some of the presented constraints will also be imposed in the handling and the ride comfort optimization analyses tackled later.

Optimization flowchart The process for the computation of the objective function, the constraints and the gradients, and their integration in the optimization flowchart, is depicted in Figure 6.3.

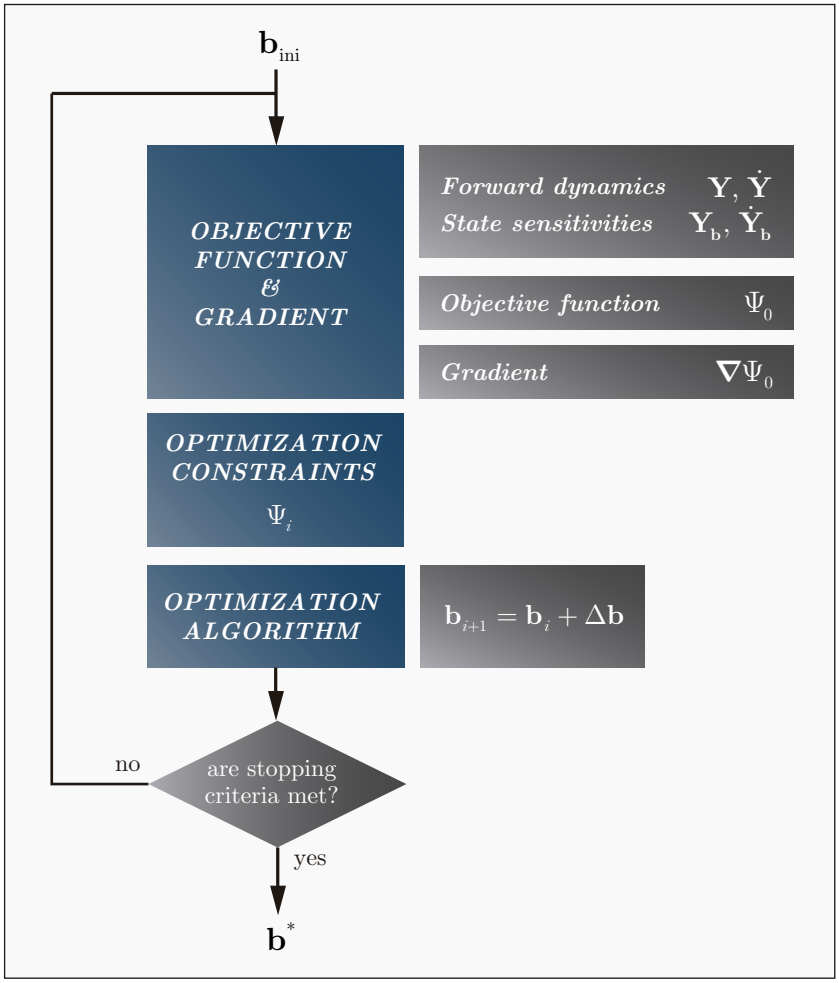


Figure 6.3: Flowchart of the optimization loop.

When both the objective function and its gradient are required by the optimization method, the sequence of operations is as follows: first, the system motion together with the state sensitivities are calculated by solving the motion differential equations augmented with direct-differentiation sensitivity equations. Second, the objective function is computed from the dynamic response. Third and last, the design sensitivity equations are solved for the objective function gradient, by using both the dynamic response variables and the state sensitivities.

A very similar procedure is followed with optimization constraints, except that, in the current state of the code, sensitivities of the optimization constraints are computed only numerically. Eventually, the implementation could be improved so that the constraint gradients could also be computed using state sensitivities.

6.2.2 Results

Three parameters were chosen for the identification: the front axle air spring stiffness, the rear axle one, and the chassis (or bodywork) torsional stiffness:

$$\mathbf{b} = \left\{k_{\text{front}}, k_{\text{rear}}, k_{\text{chassis}}\right\}^T \tag{6.25}$$

In the absence of experimental data, the nominal response of the coach (with the nominal parameters) was used as a reference, and the three selected parameters were artificially modified so as to represent the initial (estimated) design parameters. Specifically, the initial parameters were set as:

$$\mathbf{b}_{\text{ini}} = 0.6\,\mathbf{b}_{\text{nom}} \tag{6.26}$$

As far as the optimization method, MATLAB's implementation of the SQP method (`sqp`) is used to solve the objective function in Eq. (6.20), and obtain a chassis position that resembles the reference test as much as possible. Grip and hop constraints, as detailed in Eqs. (6.22) and (6.23), have been included.

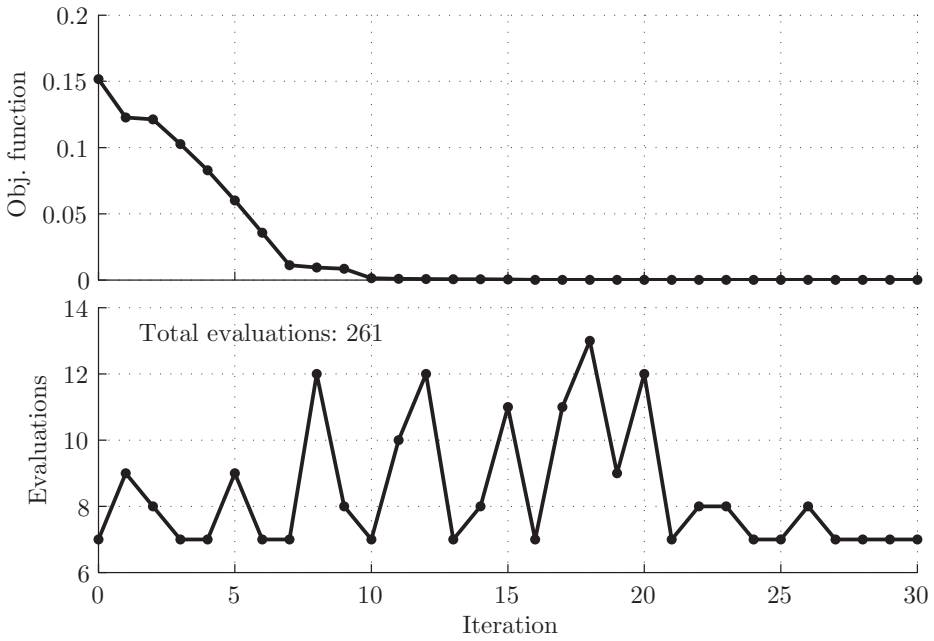


Figure 6.4: Objective function value and evaluations (parameter identification).

Figure 6.4 shows the evolution of the objective function throughout the optimization process, as well as the number of function evaluations per iteration. Figure 6.5 shows the values of the three parameters. The values converge to the reference values quite rapidly and in a stable way. Finally, Figure 6.6 gathers three different Z-acceleration curves: the original (reference) response, the initial (estimated) response and the one corresponding to the identified parameters. As clearly shown, the behavior of the vehicle with the identified parameters is exactly the same as with the reference data, as intended.

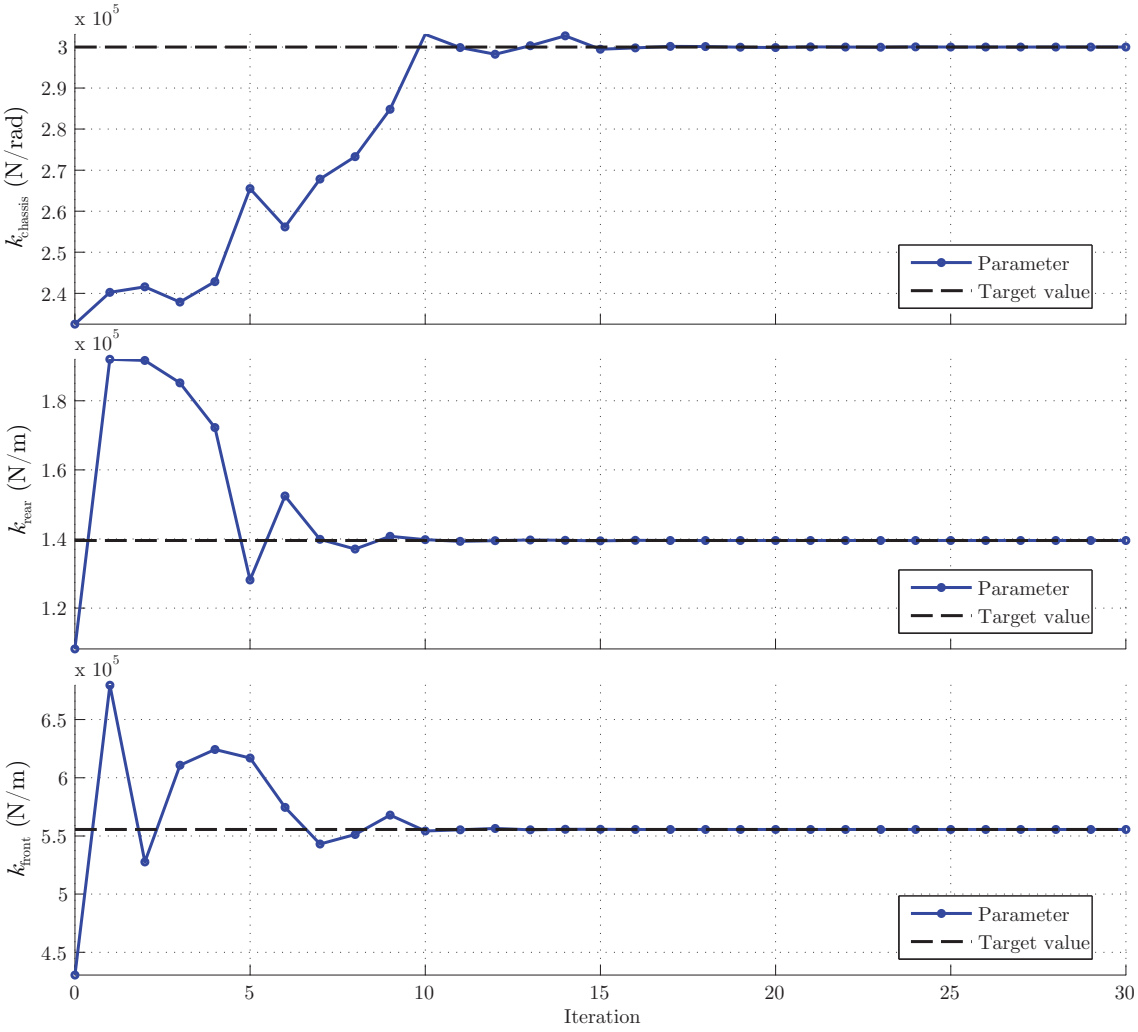


Figure 6.5: Design parameter values (parameter identification).

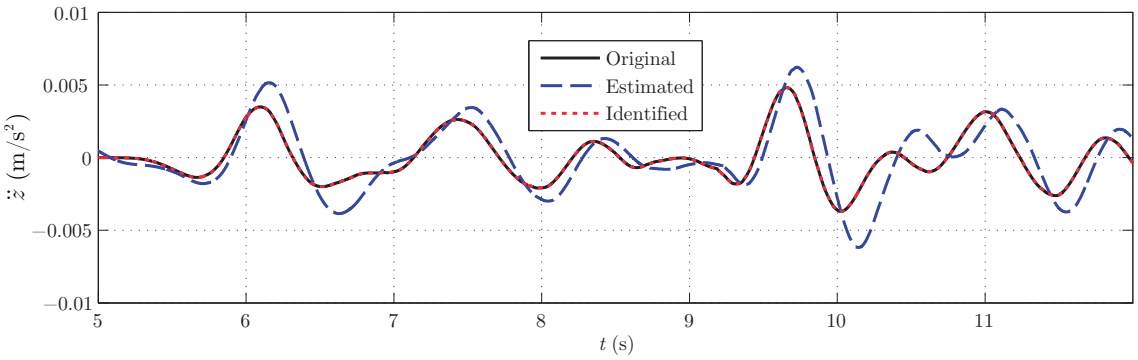


Figure 6.6: Original, estimated and identified Z-accelerations.

Even though the presented identification problem is a simplified example, it shows the potential of the optimization framework, tests the optimization constraints and the sensitivity analysis, and opens the door to more sophisticated identification and dynamic response optimization tasks.

6.3 Handling optimization

The optimization of the handling behavior is hereby carried out in light of the ideas presented in Section 3.2, about the dynamic behavior of the coach in maneuvers that stress the lateral stability of the vehicle.

6.3.1 Problem definition

First of all, note that the response optimization should be performed in the worst possible conditions. From the handling point of view, the passenger load raises the COG and causes a considerable increase of the bodywork mass (around 3 metric tons), which increases the roll angles and the load transfers. Therefore, the coach is considered loaded throughout the optimization process.

As already mentioned, the evaluation of the handling characteristics of vehicles is not a closed topic. The variety of vehicle configurations and the complexity of vehicle dynamics make it difficult to standardize results and develop regulations. Moreover, the way dynamic responses are measured for a particular objective is always subjective, and depends on the specific vehicle and purpose of the optimization. The reason for this is that results generally cannot be extended to other vehicles or situations. There are, however, a few regulations available in the literature which can help identify desirable handling characteristics: ISO 3888 (Test track for a severe lane-change manoeuvre), ISO 4138 (Steady-state circular driving behaviour — Open-loop test methods) and ISO 7401 (Lateral transient response test methods — Open-loop test methods). These maneuvers were already analyzed in Section 3.2.

Design parameters When choosing the design parameters of a real mechanical system, several aspects have to be taken into account. First, the simplicity of modification of the physical parameters in real life. For instance, changing the inertia properties of the bodywork would imply adding or removing ballast masses, which does not strike as a robust solution. Second, vehicle manufacturers usually have practical manufacturing, assembling and geometrical constraints that limit the choice of design parameters. For example, changing the geometry of the rear axle would imply long-term development changes that cannot be easily integrated in the production process. For these reasons, the potential of the sensitivity analysis developed in Chapter 5 is hereby going to be limited to the analysis of stiffness and damping suspension properties, both for the handling and the ride comfort studies. Figure 6.1 shows the list of selected suspension parameters.

#	Name	Initial value	Lower bound	Upper bound	Units
1	Front spring stiffness	$5.56\times10^{+5}$	$1.39\times10^{+5}$	$1.11\times10^{+6}$	N/m
2	Front spring preload	2.35×10^{-1}	5.86×10^{-2}	4.69×10^{-1}	m
3	Front damper c_1 coefficient	$1.98\times10^{+4}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
4	Front damper c_2 coefficient	$9.92\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
5	Front damper c_3 coefficient	$9.92\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
6	Front damper c_4 coefficient	$4.96\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
7	Front damper v_{11} coefficient	1.00×10^{-2}	$0.00\times10^{+0}$	1.00×10^{-1}	m/s
8	Front damper v_{22} coefficient	-1.00×10^{-2}	-1.00×10^{-1}	$0.00\times10^{+0}$	m/s
9	Rear spring stiffness	$1.40\times10^{+5}$	$3.49\times10^{+4}$	$2.79\times10^{+5}$	N/m
10	Rear spring preload	3.08×10^{-1}	7.69×10^{-2}	6.15×10^{-1}	m
11	Rear damper c_1 coefficient	$1.98\times10^{+4}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
12	Rear damper c_2 coefficient	$9.92\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
13	Rear damper c_3 coefficient	$9.92\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
14	Rear damper c_4 coefficient	$4.96\times10^{+3}$	$0.00\times10^{+0}$	$1.00\times10^{+5}$	N·s/m
15	Rear damper v_{11} coefficient	1.00×10^{-2}	$0.00\times10^{+0}$	1.00×10^{-1}	m/s
16	Rear damper v_{22} coefficient	-1.00×10^{-2}	-1.00×10^{-1}	$0.00\times10^{+0}$	m/s
17	Front anti-roll bar stiffness	$2.16\times10^{+5}$	$5.40\times10^{+4}$	$4.32\times10^{+5}$	N·m/rad
18	Rear anti-roll bar stiffness	$2.00\times10^{+5}$	$5.00\times10^{+4}$	$4.00\times10^{+5}$	N·m/rad
19	Bodywork torsion stiffness	$3.00\times10^{+5}$	$7.50\times10^{+4}$	$6.00\times10^{+5}$	N·m/rad

Table 6.1: Design parameters for the suspension design.

Objective functions As already explained, the main objective of handling is to improve road-holding properties. The coach response has already been studied in three different maneuvers: the step-steer test (SS), the constant speed test (CS) and the double lane-change maneuver (DLC). Now, suitable measurements (or metrics) of the vehicle stability have to be defined, so that the minimization of the corresponding objective function leads to a consistent improvement of the handling characteristics.

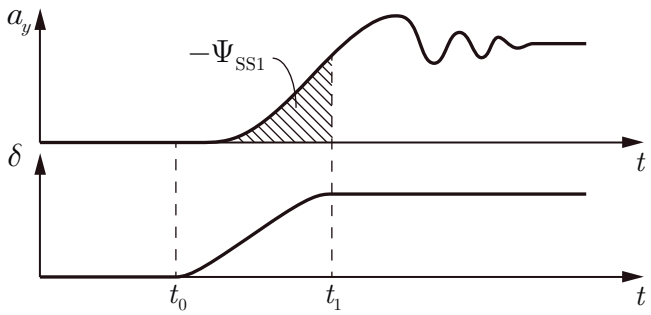


Figure 6.7: Steering wheel input and lateral acceleration in the step-steer test.

In the step-steer test, the lateral acceleration is chosen as the monitored magnitude. The sooner the vehicle responds to the steering input, the better the vehicle is going to follow the driver's commands. In terms of the lateral acceleration, the sooner the lateral acceleration starts to change, the better. One possible way of formulating this idea over time is to compute the definite integral of the lateral acceleration during the transient steering interval:

$$\Psi_{\text{SS1}} = -\int_{t_0}^{t_1} a_y(t, \ddot{\mathbf{z}}, \mathbf{b}) dt \tag{6.27}$$

where t_0 and t_1 are depicted in Figure 6.7. Differentiating this expression with respect to the design parameters, the gradient of the objective function can be written in terms of the state sensitivities. Assuming that Eq. (3.21) holds, the following expression for the objective gradient can be written:

$$\frac{d\Psi_{\text{SS1}}}{d\mathbf{b}} = -\int_{t_0}^{t_1} \frac{da_y}{d\mathbf{b}} dt = -\int_{t_0}^{t_1} \frac{1}{2\sqrt{\ddot{x}^2 + \ddot{y}^2}} \left(2\ddot{x} \frac{d\ddot{x}}{d\mathbf{b}} + 2\ddot{y} \frac{d\ddot{y}}{d\mathbf{b}} \right) dt \tag{6.28}$$

which can be solved using the state sensitivities.

A second metric for the minimization of handling instability is also related to the lateral acceleration response. This time, the goal is to reduce the peak value of the lateral acceleration with respect to its final (steady-state) value. This objective can be written as:

$$\Psi_{\text{SS2}} = \sqrt{\int_{t_1}^{t_f} \left[a_y(t, \ddot{\mathbf{z}}, \mathbf{b}) - a_y(t_f) \right]^2 dt} \tag{6.29}$$

where the time discretization is the same as in the previous objective function. This equation minimizes the amplitude of the oscillations from the moment the steering input becomes stable onwards. Taking derivatives:

$$\begin{aligned} 2\Psi_{\text{SS2}} \frac{d\Psi_{\text{SS2}}}{d\mathbf{b}} &= \int_{t_1}^{t_f} 2\left(\overline{a}_y - \overline{a}_y^f \right) \left(\frac{da_y}{d\mathbf{b}} - \frac{da_y^f}{d\mathbf{b}} \right) dt \\ \overline{a}_y &\equiv \frac{1}{2\sqrt{\ddot{x}^2 + \ddot{y}^2}} \left(2\ddot{x} \frac{d\ddot{x}}{d\mathbf{b}} + 2\ddot{y} \frac{d\ddot{y}}{d\mathbf{b}} \right) \\ \overline{a}_y^f &\equiv \frac{1}{2\sqrt{\ddot{x}^{f2} + \ddot{y}^{f2}}} \left(2\ddot{x}^f \frac{d\ddot{x}^f}{d\mathbf{b}} + 2\ddot{y}^f \frac{d\ddot{y}^f}{d\mathbf{b}} \right) \end{aligned} \tag{6.30}$$

from which the corresponding derivatives w.r.t. the parameters can be found.

The second maneuver used as a metric of handling behavior is the very well-known double lane-change maneuver, already studied in Section 3.2.2. In general, an excessive bodywork roll is harmful for lateral stability, because it increases the load transfers and can reduce tire grip (see Section 3.2.1). Thus, it would be desirable to reduce the roll angles. To that end, two different formulas are used. The first one minimizes the bodywork roll through the RMS value:

$$\Psi_{\text{DLC1}} = \sqrt{\int_{t_i}^{t_f} \varphi^2(t, \mathbf{b})\, dt} \tag{6.31}$$

Again, taking derivatives w.r.t. the design parameters to compute sensitivities:

$$2\Psi_{\text{DLC1}} \frac{d\Psi_{\text{DLC1}}}{d\mathbf{b}} = \int_{t_i}^{t_f} 2\varphi \frac{d\varphi}{d\mathbf{b}} dt \tag{6.32}$$

The second objective function is quite similar to the first one, but the roll velocity is minimized instead:

$$\Psi_{\text{DLC2}} = \sqrt{\int_{t_i}^{t_f} \dot{\varphi}^2(t, \mathbf{b})\, dt} \tag{6.33}$$

and thus

$$2\Psi_{\text{DLC2}} \frac{d\Psi_{\text{DLC2}}}{d\mathbf{b}} = \int_{t_i}^{t_f} 2\dot{\varphi} \frac{d\dot{\varphi}}{d\mathbf{b}} dt \tag{6.34}$$

The last maneuver for the assessment of the handling quality is the constant speed test, already described in Section 3.2.4. At very low speeds, the path radius is determined by the geometry of the suspension system, specifically by the wheel base and the front and rear wheel steer angles, which is referred to as the Ackermann condition. When the speed increases, centrifugal forces influence the front and rear axles differently, modifying the relationship between the steering angle and the path radius. According to ISO 4138 standard, the difference between the front and rear cornering compliance is defined as the understeering gradient (degrees per meter per second squared), which is here used as a metric for the improvement of the response. A minimization of the understeering gradient would be desirable, which can be expressed as the following RMS value:

$$\Psi_{\text{CS}} = \sqrt{\int_{t_i}^{t_f} \left[\frac{d\delta}{da_y}(t, \mathbf{z}, \ddot{\mathbf{z}}, \mathbf{b})\right]^2 dt} = \sqrt{\int_{t_i}^{t_f} \left[\frac{da_y}{d\delta}(t, \mathbf{z}, \ddot{\mathbf{z}}, \mathbf{b})\right]^{-2} dt} \tag{6.35}$$

The objective function can then be differentiated w.r.t. the parameters to find the state sensitivities, as follows:

$$2\Psi_{\text{CS}} \frac{d\Psi_{\text{CS}}}{d\mathbf{b}} = -\int_{t_i}^{t_f} 2\left(\frac{da_y}{d\delta}\right)^{-3} \frac{d}{d\mathbf{b}}\left(\frac{da_y}{d\delta}\right) dt \tag{6.36}$$

and by applying Schwarz's theorem and expanding the lateral acceleration term:

$$\begin{aligned} \Psi_{\text{CS}} \frac{d\Psi_{\text{CS}}}{d\mathbf{b}} &= -\int_{t_i}^{t_f} \left(\frac{da_y}{d\delta}\right)^{-3} \frac{d}{d\delta}\left(\frac{da_y}{d\mathbf{b}}\right) dt = \\ &= -\int_{t_i}^{t_f} \left(\frac{da_y}{d\delta}\right)^{-3} \frac{d}{d\delta}\left(\frac{1}{2\sqrt{\ddot{x}^2 + \ddot{y}^2}}\left(2\ddot{x}\frac{d\ddot{x}}{d\mathbf{b}} + 2\ddot{y}\frac{d\ddot{y}}{d\mathbf{b}}\right)\right) dt \end{aligned} \tag{6.37}$$

which is the final expression for the computation of design sensitivities. Even though the design and state sensitivities of this maneuver have been computed successfully, this objective function is not included in the global handling objective function for two reasons: first, the influence of the chosen parameters on the objective is very little; second, the time integration required for this maneuver is almost three times the integration time of the others, and thus has proven to be untimely for the purpose of efficient optimization. Nevertheless, it has been included here for the sake of theoretical completeness.

The handling objective functions are combined into a single handling objective function using a scalarization technique. The handling functions, Ψ_k, are weighted and added to an accumulated objective function, Ψ_h^{Σ}. The importance of each objective function is relative to the unit objective function reduction that it would achieve on its own:

$$\Psi_h^{\Sigma}(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) = \sum_k w_k \left(\frac{\Psi_k(\mathbf{b}) - \Psi_k^*}{\Psi_k^*} \right)^2 \qquad (6.38)$$
$$k = \text{SS1, SS2, DLC1, DLC2}$$

where $\Psi_k^* = \Psi_k(\mathbf{b}^*)$ is the objective function value accomplished when only Ψ_k is minimized. The weighting coefficients, w_k, are computed from:

$$w_k \left(\frac{\Psi_k(\mathbf{b}_{\text{ini}}) - \Psi_k^*}{\Psi_k^*} \right)^2 = \frac{1}{4} \qquad (6.39)$$

The same strategy will be followed with ride comfort objective functions in the next subsection. Table 6.2 shows the final objective function values of the independent optimization procedures.

Ψ_k	$\Psi_k(\mathbf{b}_{\text{ini}})$	$\Psi_k(\mathbf{b}^*)$	Improvement	w_k
Ψ_{SS1}	–0.812	–0.855	-5%	97.567
Ψ_{SS2}	1.033	0.309	70%	0.046
Ψ_{DLC1}	0.025	0.022	11%	15.016
Ψ_{DLC2}	0.046	0.036	20%	3.815

Table 6.2: Independent objective function values (handling).

Optimization constraints As already explained in the parameter identification problem, the dynamic response optimization of vehicles requires a careful definition of optimization constraints. In the case of handling optimization, the constraints presented in the parameter identification problem (Section 6.2), are included, namely the grip constraints ($\mathbf{\Psi}^{\text{grip}}$), the hop constraints ($\mathbf{\Psi}^{\text{hop}}$), and the box constraints ($\mathbf{\Psi}^{\text{box}}$). The mathematical expressions of the first two types can be found in Eqs. (6.22) and (6.23), respectively. In addition to these, the

damper design parameters need additional constraints to ensure their relationship, according to Figure 3.10:

$$\boldsymbol{\Psi}^{\mathrm{dam}} \equiv \begin{Bmatrix} c_2 - c_1 \\ c_4 - c_3 \\ c_3 - c_1 \\ -v_{11} \\ v_{22} \end{Bmatrix} \leq \mathbf{0} \tag{6.40}$$

The two front dampers have the same characteristics, and thus need only one set of damper constraints, $\boldsymbol{\Psi}^{\mathrm{dam}}$. Similarly, the rear dampers share one set of damper constraints. In total, the number of optimization constraints is 22 (6 grip constraints, 6 hop constraints, 5 front damper constraints, 5 rear damper constraints), plus 38 box constraints (two per design parameter), which are handled independently by MATLAB's optimization methods.

During the optimization process, there is a chance for the box constraints to be violated. This happens generally for two reasons. First, during finite differencing, an out-of-range value is picked. The reason for this is that the numerical perturbation is somewhat "blind". Second, when an excessive perturbation of the parameters causes an erratic system behavior that invalidates the evaluation of the objective function and/or the constraints. In this case, the objective function would not return a real value (but rather a `NaN`, for example). Both phenomena are harmful for the optimization process, because they can lead to infeasible points, and thus should be detected and prevented.

6.3.2 Results

The coach suspension system has been optimized for good handling behavior by defining the design parameters in Table 6.1, the objective function in Eq. (6.38) and the optimization constraints just presented. In short, the optimization problem can be expressed as:

$$\begin{aligned} &\min\ \Psi_h^{\Sigma}(t, \mathbf{z}, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) \\ &\text{s.t.}\ \left\{ \boldsymbol{\Psi}^{\mathrm{grip}\,T}, \boldsymbol{\Psi}^{\mathrm{hop}\,T}, \boldsymbol{\Psi}^{\mathrm{dam}\,T}, \boldsymbol{\Psi}^{\mathrm{box}\,T} \right\}^T \end{aligned} \tag{6.41}$$

On every iteration of the optimization algorithm, once the system motion and the state sensitivities are computed, objective functions Ψ_{SS1}, Ψ_{SS2}, Ψ_{CS}, Ψ_{DLC1}, Ψ_{DLC2} are evaluated and combined into Ψ_h^{Σ}. If the objective function gradient is required, the corresponding expressions are solved, as are the optimization constraints. Regarding optimization methods, the following algorithms have been used for the handling problem and later for the comfort problem:

– `fmincon`: `sqp`, `interior-point`
– `fminunc`: `trust-region`, `quasi-newton`
– `gaoptimset`: `genetic-algorithm`

which correspond to gradient-based, penalty and global methods, respectively. However, for the sake of conciseness, only results with the `fminunc/sqp` algorithm are presented here. A comparison between optimization methods for a single objective function will be presented in Section 6.6.

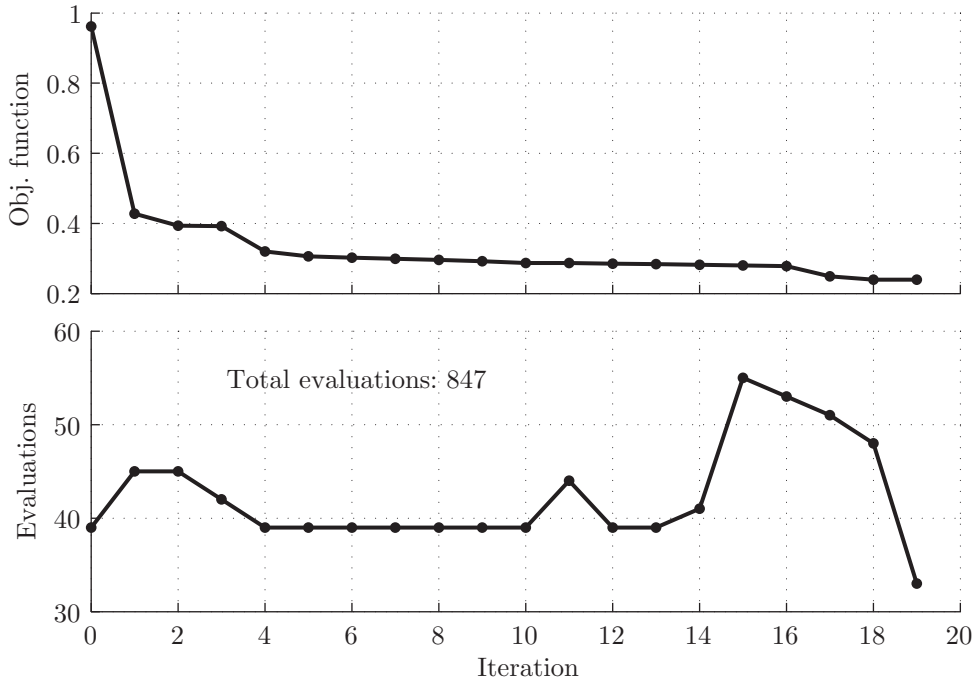


Figure 6.8: Objective function value and iterations (handling).

The objective function value during the optimization process and the number of evaluations per iteration are shown in Figure 6.8. In accordance with Eqs. (6.38) and (6.39), the accumulated objective function starts with a value of 1.0 and decreases significantly until close to 0.2, meaning that the four individual objective functions are consistent and can be reduced simultaneously.

On the other hand, the normalized values of the design parameters are shown in Figure 6.9, where the parameters are sorted as in Table 6.1 (descriptive labels are kept to facilitate the reading). Only one parameter appears to be reaching the box constraint limit, which means the parameter limits and the constraints were well established. The true values of the parameters in the initial and optimized stages are gathered in Table 6.3.

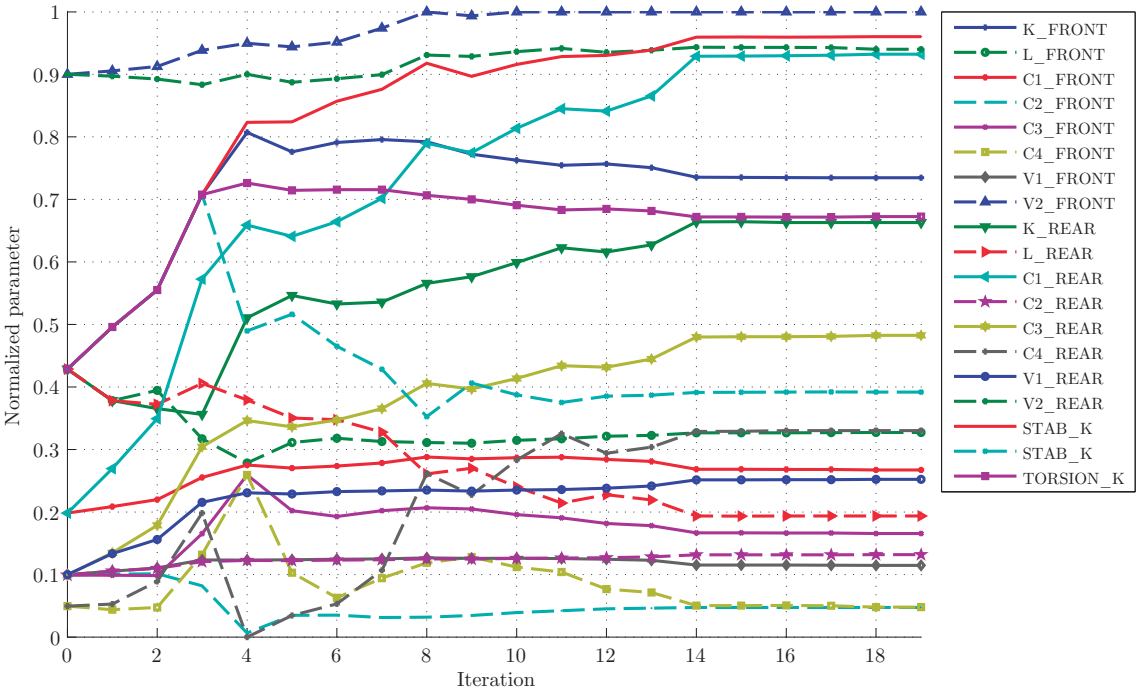


Figure 6.9: Normalized design parameter values (handling).

A few dynamic responses are checked before and after the optimization, in order to assess the improvement of the handling behavior. First, the roll acceleration and the load transfers are evaluated in the step-steer maneuver (Figure 6.10). It is clearly shown how the roll acceleration amplitude is greatly reduced in the optimized setup. Also, the steering response is faster, which, as explained before, is a handling quality. Finally, both front and rear load transfers are reduced significantly, which is also a very desirable quality for vehicle maneuverability.

Second, the roll angle and the load transfers are checked in the double lane-change maneuver (see Figure 6.11). The roll angle amplitude is reduced with the optimized suspension setup. Again, the roll response is faster too. Finally, the load transfers are reduced, which improves the tire grip. Thus, from an analytical point of view, the dynamic response has been improved in terms of the defined maneuvers and objective functions.

Objective or parameter	Initial value	Final value
Ψ_h^{Σ}	$1.00\times10^{+0}$	2.39×10^{-1}
1	$5.56\times10^{+5}$	$8.53\times10^{+5}$
2	2.35×10^{-1}	1.93×10^{-1}
3	$1.98\times10^{+4}$	$2.67\times10^{+4}$
4	$9.92\times10^{+3}$	$4.74\times10^{+3}$
5	$9.92\times10^{+3}$	$1.66\times10^{+4}$
6	$4.96\times10^{+3}$	$4.80\times10^{+3}$
7	1.00×10^{-2}	1.15×10^{-2}
8	-1.00×10^{-2}	-7.85×10^{-13}
9	$1.40\times10^{+5}$	$1.97\times10^{+5}$
10	3.08×10^{-1}	1.81×10^{-1}
11	$1.98\times10^{+4}$	$9.32\times10^{+4}$
12	$9.92\times10^{+3}$	$1.32\times10^{+4}$
13	$9.92\times10^{+3}$	$4.83\times10^{+4}$
14	$4.96\times10^{+3}$	$3.30\times10^{+4}$
15	1.00×10^{-2}	2.52×10^{-2}
16	-1.00×10^{-2}	-5.99×10^{-3}
17	$2.16\times10^{+5}$	$4.17\times10^{+5}$
18	$2.00\times10^{+5}$	$1.87\times10^{+5}$
19	$3.00\times10^{+5}$	$4.28\times10^{+5}$

Table 6.3: Final objective function and parameter values (handling).

Finally, the responses are compared by simulating the initial vehicle and the optimized vehicle simultaneously, rendering the results in the graphics viewer described in Section 2.5. Figure 6.12 shows three different images: the step-steer test, the double lane-change maneuver and the constant speed test, where the original suspension is depicted in blue and the optimized one in red. In the first image, the fact that the optimized suspension leans further towards the center of the curve, means that it reacts faster to the left-turn steering input. In the second, the front triangles of the optimized suspension are in a more horizontal position, meaning that the load transfer and the roll angle are smaller. The third and last image corresponds to the constant speed test, in which, after a very long left turn, the optimized vehicle position is outside the original one, indicating that the coach has a smaller overturning behavior. Additional points of view and videos can further demonstrate these concepts.

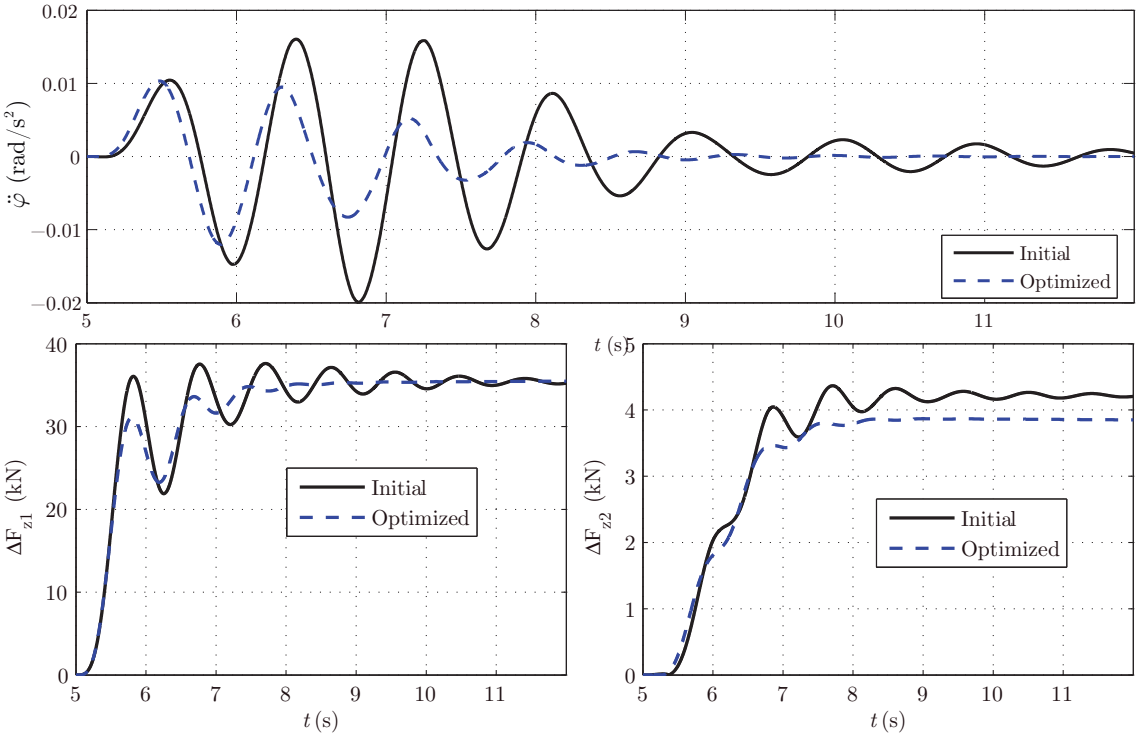


Figure 6.10: Comparison between initial and optimized responses (step-steer).

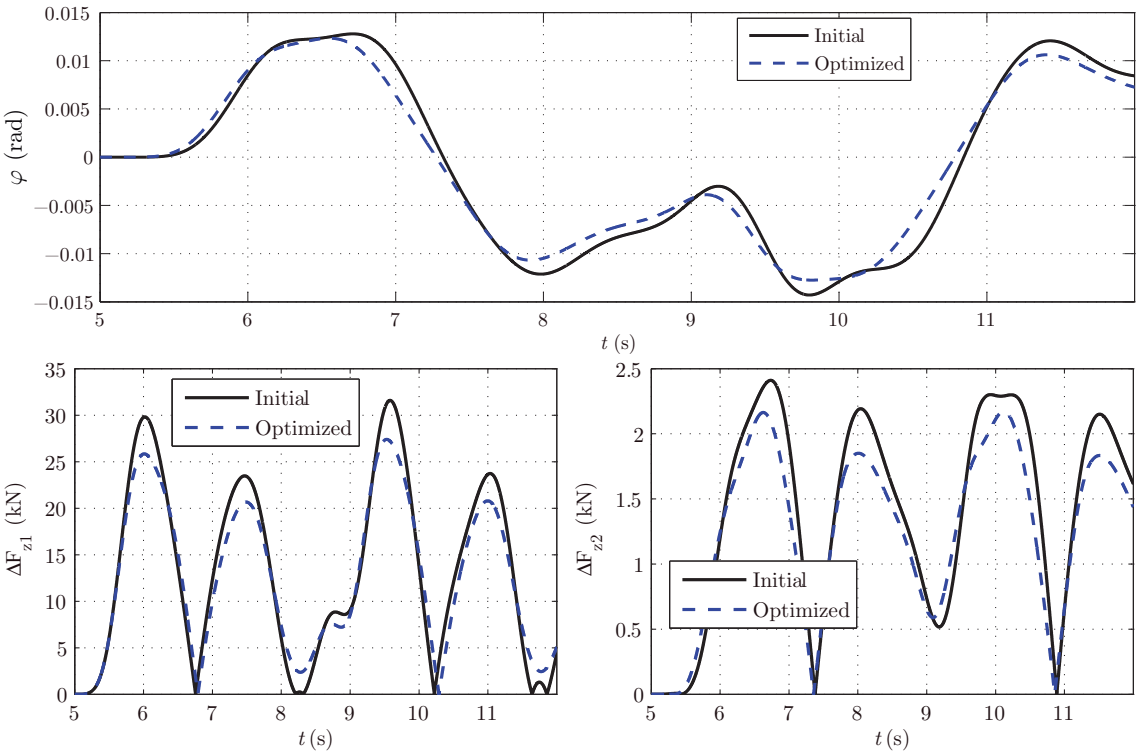


Figure 6.11: Comparison between initial and optimized responses (double lane-change).

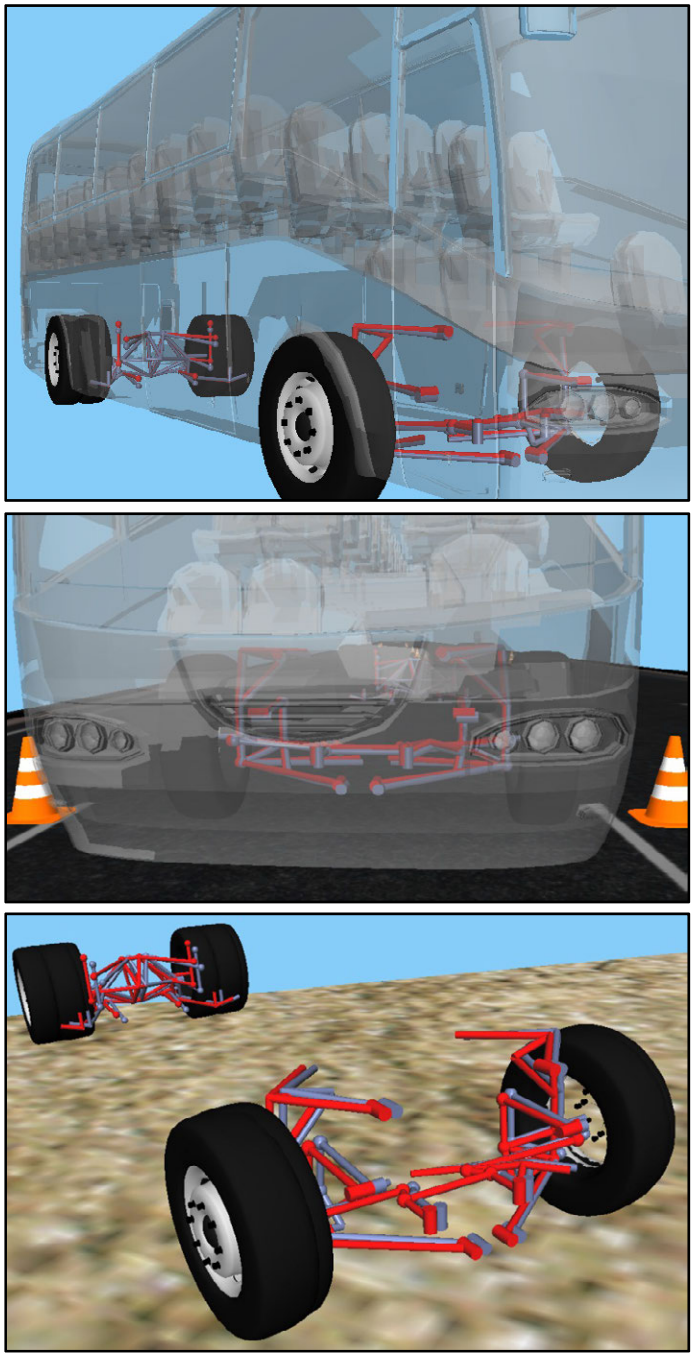


Figure 6.12: 3D comparison between original and optimized suspensions (handling). Top: step-steer test. Middle: double lane-change. Bottom: constant speed test.

6.4 Ride comfort optimization

Similarly to the dynamic response optimization, the ride comfort foundations can be found in Chapter 3. Specifically, Section 3.3 analyzes the ride comfort response of the coach model.

6.4.1 Problem definition

Comfort is usually considered as an opposition to handling, i.e., the improvement of the handling response usually implies a worsening of comfort characteristics. Therefore, the design of suspension systems requires a trade-off between handling and comfort responses. Methods for measuring vibrations can be complex, and the effect of vibration on the comfort and health of the human body is difficult to assess (Griffin, 2007). In spite of this, some criteria have been gathered in international standards, most importantly in ISO 2631 standard. Also, authors agree that vibration is most accurately evaluated looking at velocity and acceleration effects on the passengers.

Design parameters The design parameters considered in the ride comfort optimization are exactly the same ones as in the handling analysis (Table 6.1).

Objective functions Two different maneuvers are analyzed, as possible scenarios in which vibration discomfort becomes more acute: the speed bumps test and the four-post test (see Section 3.3). Also, several ways of adding up accelerations over time can be considered (mean value, RMS value, VDV, etc.). Accordingly, different ways of formulating ride comfort objective functions result. In all of them, the objective is to obtain a single scalar that quantifies the amount of discomfort caused by the maneuver.

In the speed bumps test, the RMS value of the *Z*-velocity of the bodywork is selected in the first place:

$$\Psi_{\text{BMP1}} = \sqrt{\int_{t_i}^{t_f} \dot{z}^2(t,\mathbf{b})\,dt} \tag{6.42}$$

$$\frac{d\Psi_{\text{BMP1}}}{d\mathbf{b}} = \frac{1}{\Psi_{\text{BMP1}}} \int_{t_i}^{t_f} \dot{z}\,\frac{d\dot{z}}{d\mathbf{b}}\,dt \tag{6.43}$$

where the objective function has been differentiated w.r.t. the parameters to obtain an expression for the gradient in terms of the state sensitivities. Secondly, the RMS value is calculated upon the *Z*-acceleration to arrive at the second objective function and its corresponding gradient:

$$\Psi_{\text{BMP2}} = \sqrt{\int_{t_i}^{t_f} \ddot{z}^2(t,\mathbf{b})\,dt} \tag{6.44}$$

$$\frac{d\Psi_{\text{BMP2}}}{d\mathbf{b}} = \frac{1}{\Psi_{\text{BMP2}}} \int_{t_i}^{t_f} \ddot{z}\,\frac{d\ddot{z}}{d\mathbf{b}}\,dt \tag{6.45}$$

As far as the four-post test is concerned, very similar objective functions are defined, except the VDV value is used instead of the RMS value. The first expression would be:

$$\Psi_{\text{PRF1}} = \left(\int_{t_i}^{t_f} \dot{z}^4(t, \mathbf{b})\, dt \right)^{1/4} \tag{6.46}$$

$$\frac{d\Psi_{\text{PRF1}}}{d\mathbf{b}} = \frac{1}{\Psi_{\text{PRF1}}^3} \int_{t_i}^{t_f} \dot{z}^3 \frac{d\dot{z}}{d\mathbf{b}}\, dt \tag{6.47}$$

and the second one:

$$\Psi_{\text{PRF2}} = \left(\int_{t_i}^{t_f} \ddot{z}^4(t, \mathbf{b})\, dt \right)^{1/4} \tag{6.48}$$

$$\frac{d\Psi_{\text{PRF2}}}{d\mathbf{b}} = \frac{1}{\Psi_{\text{PRF2}}^3} \int_{t_i}^{t_f} \ddot{z}^3 \frac{d\ddot{z}}{d\mathbf{b}}\, dt \tag{6.49}$$

Finally, the ride comfort objective functions are combined into a single objective function, as done before with the handling optimization:

$$\Psi_c^{\Sigma}(t, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) = \sum_k w_k \left(\frac{\Psi_k(\mathbf{b}) - \Psi_k^*}{\Psi_k^*} \right)^2 \tag{6.50}$$
$$k = \text{BMP1, BMP2, PRF1, PRF2}$$

where $\Psi_k^* = \Psi_k(\mathbf{b}^*)$ is the objective function value obtained when only Ψ_k is minimized, and the weighting coefficients, w_k, are computed from:

$$w_k \left(\frac{\Psi_k(\mathbf{b}_{\text{ini}}) - \Psi_k^*}{\Psi_k^*} \right)^2 = \frac{1}{4} \tag{6.51}$$

Table 6.4 shows the values of the independent objective functions, which are then used to put together the accumulated objective function.

Ψ_k	$\Psi_k(\mathbf{b}_{\text{ini}})$	$\Psi_k(\mathbf{b}^*)$	Improvement	w_k
Ψ_{BMP1}	0.106	0.071	33%	1.046
Ψ_{BMP2}	3.069	2.054	33%	1.024
Ψ_{PRF1}	0.062	0.042	33%	1.061
Ψ_{PRF2}	1.410	1.143	19%	4.582

Table 6.4: Independent objective function values (comfort).

Optimization constraints Both the speed bumps test and the four-post test generate a vertical response in the vehicle, which means that the lateral behavior has little relevance. For this reason, tire grip constraints ($\mathbf{\Psi}^{\text{grip}}$) are not included. On the other hand, there is no guarantee that the tires are going to stay in contact with the ground when sudden vibrations are input to the wheels, and therefore tire hop constraints ($\mathbf{\Psi}^{\text{hop}}$) are not included either. Only box constraints ($\mathbf{\Psi}^{\text{box}}$) and damper constraints ($\mathbf{\Psi}^{\text{dam}}$) are considered.

6.4.2 Results

The coach suspension system has been optimized for a good ride comfort behavior by defining the design parameters in Table 6.1, the objective function in Eq. (6.50) and the recently presented optimization constraints, in a way similar to the procedure carried out in the handling optimization problem. A summary of the ride comfort optimization problem would be:

$$\begin{aligned} &\min \Psi_c^{\Sigma}(t, \dot{\mathbf{z}}, \ddot{\mathbf{z}}, \mathbf{b}) \\ &\text{s.t. } \left\{ \boldsymbol{\Psi}^{\mathrm{dam}\,T}, \boldsymbol{\Psi}^{\mathrm{box}\,T} \right\}^T \end{aligned} \tag{6.52}$$

As already explained, several optimization methods have been used, but only the results corresponding to `fmincon/sqp` are presented here.

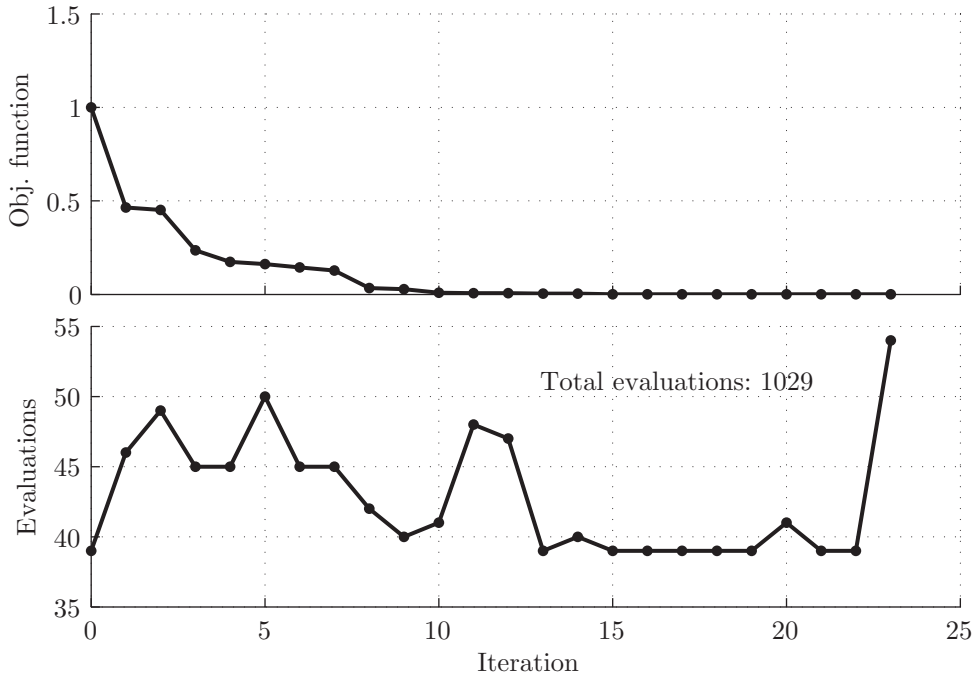


Figure 6.13: Objective function value and iterations (comfort).

The objective function value during the optimization process and the number of evaluations per iteration are shown in Figure 6.13. As happened before in the handling case, the accumulated objective function starts with a value of 1.0 and decreases significantly until very close to 0.0 (recall Eqs. (6.50) and (6.51)), meaning that the four individual objective functions are consistent and can be reduced simultaneously.

The normalized values of the design parameters are shown in Figure 6.14, where the design parameters (with descriptive labels) are sorted as done previously in Table 6.1. Only two parameters appear to be reaching the lower box constraint limit. The true values of the parameters in the initial and optimized stages are gathered in Figure 6.5.

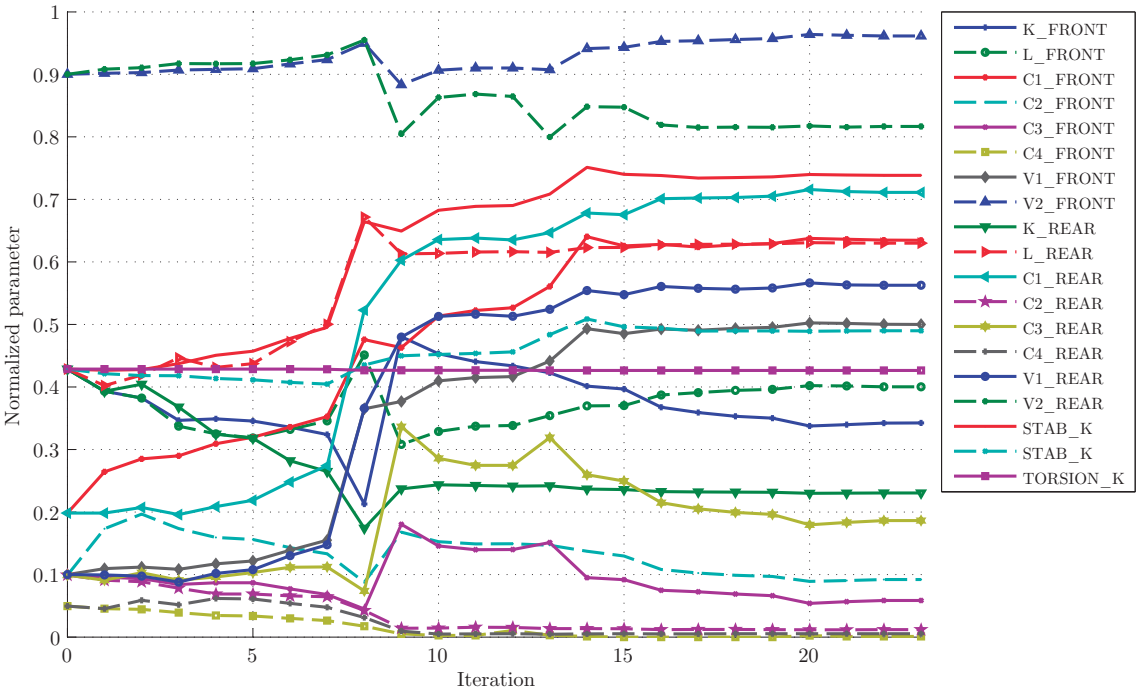


Figure 6.14: Normalized design parameter values (comfort).

Four dynamic responses have been evaluated to assess the efficacy of the comfort optimization procedure. The first two curves are the vertical acceleration and velocity of the coach bodywork in the speed bumps test (see Figure 6.15). The amplitude of both magnitudes is greatly reduced, achieving the objective sought in the optimization. A reduction in the bodywork acceleration is expected to improve passenger comfort. Secondly, the same two responses are plotted in the four-post test, as shown in Figure 6.16. A very similar improvement is observed, both in the acceleration and velocity responses. Therefore, it can be said that the comfort optimization solution is physically meaningful and not just mathematically feasible.

Finally, as done before in the handling optimization section, additional graphical proof of the behavior improvement is provided. Two maneuver screenshots are shown in Figure 6.17, where the optimized vehicle (in red) is superimposed on the original one (in blue). The first one corresponds to one of the moments of the speed bump test, where the chassis is bouncing down, and clearly shows the original vehicle at a lower position, i.e., with a larger oscillation amplitude. The second screenshot is taken in the four-post test, and shows that the roll angle of the rear support is smaller in the optimized vehicle than in the original one, which is obviously better for the comfort experience. These differences can be better assessed from additional points of view and by looking at the complete maneuver animation.

Objective or parameter	Initial value	Final value
Ψ_c^{Σ}	$1.00\times10^{+0}$	2.50×10^{-6}
1	$5.56\times10^{+5}$	$4.72\times10^{+5}$
2	2.35×10^{-1}	2.23×10^{-1}
3	$1.98\times10^{+4}$	$6.35\times10^{+4}$
4	$9.92\times10^{+3}$	$9.19\times10^{+3}$
5	$9.92\times10^{+3}$	$5.85\times10^{+3}$
6	$4.96\times10^{+3}$	$9.96\times10^{+1}$
7	1.00×10^{-2}	5.00×10^{-2}
8	-1.00×10^{-2}	-3.85×10^{-3}
9	$1.40\times10^{+5}$	$9.12\times10^{+4}$
10	3.08×10^{-1}	4.16×10^{-1}
11	$1.98\times10^{+4}$	$7.11\times10^{+4}$
12	$9.92\times10^{+3}$	$1.17\times10^{+3}$
13	$9.92\times10^{+3}$	$1.86\times10^{+4}$
14	$4.96\times10^{+3}$	$5.32\times10^{+2}$
15	1.00×10^{-2}	5.63×10^{-2}
16	-1.00×10^{-2}	-1.83×10^{-2}
17	$2.16\times10^{+5}$	$3.33\times10^{+5}$
18	$2.00\times10^{+5}$	$2.21\times10^{+5}$
19	$3.00\times10^{+5}$	$2.99\times10^{+5}$

Table 6.5: Final objective function and parameter values (comfort).

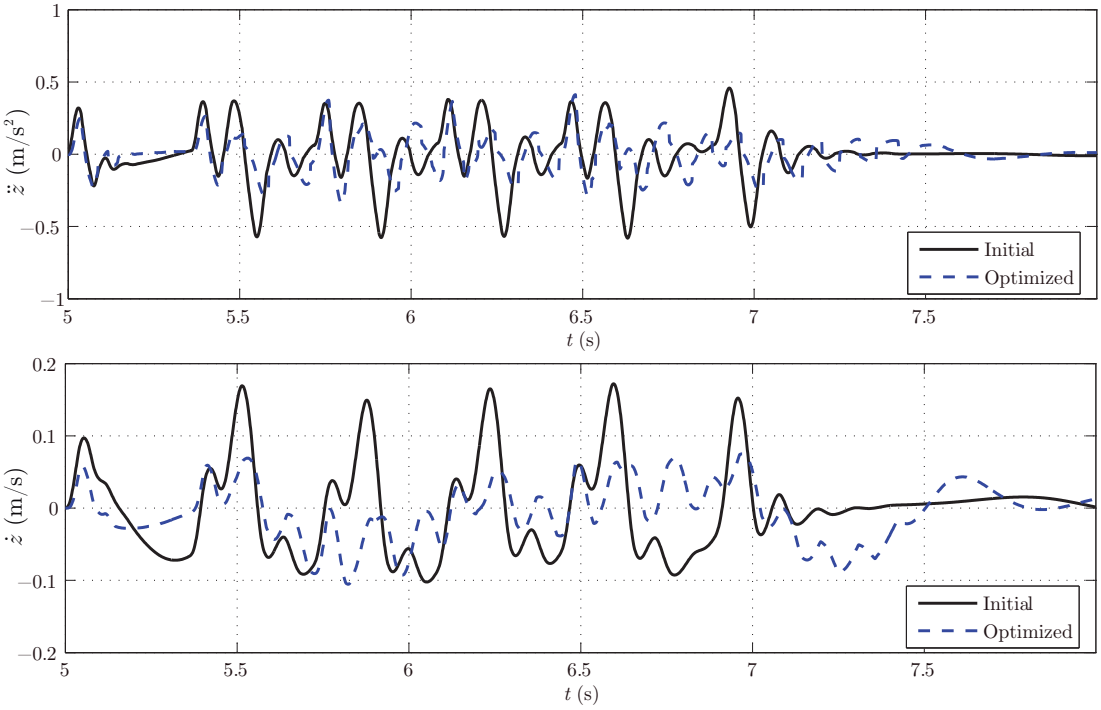


Figure 6.15: Comparison between initial and optimized responses (speed bumps).

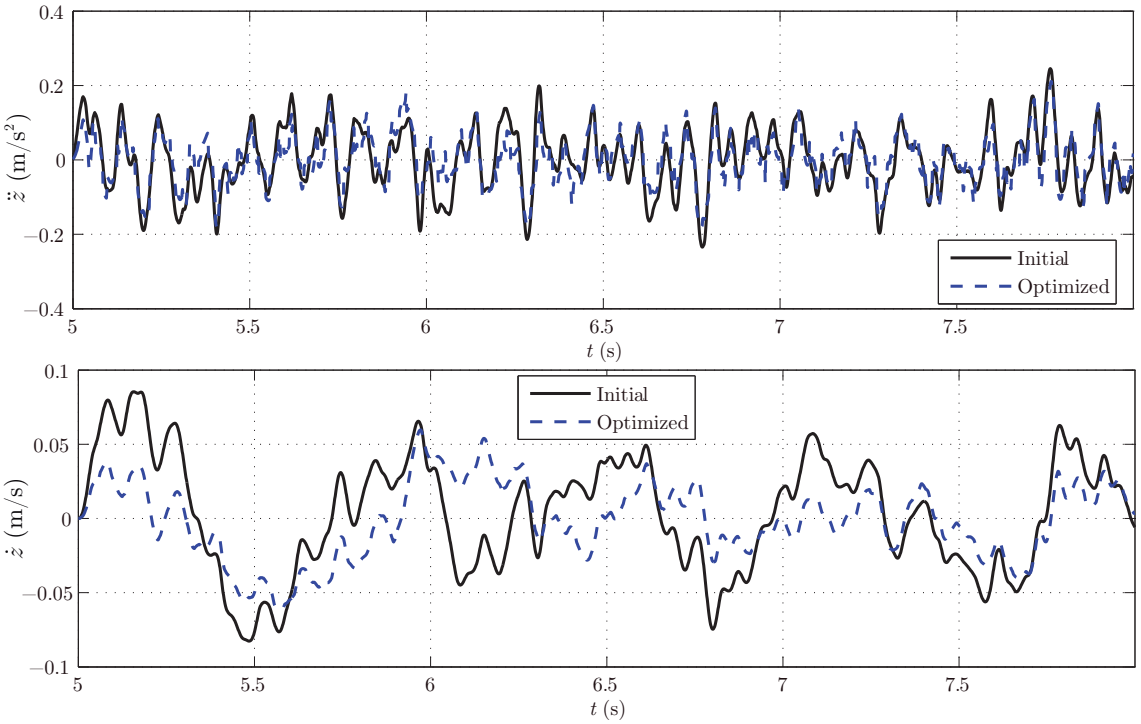


Figure 6.16: Comparison between initial and optimized responses (four-post test).

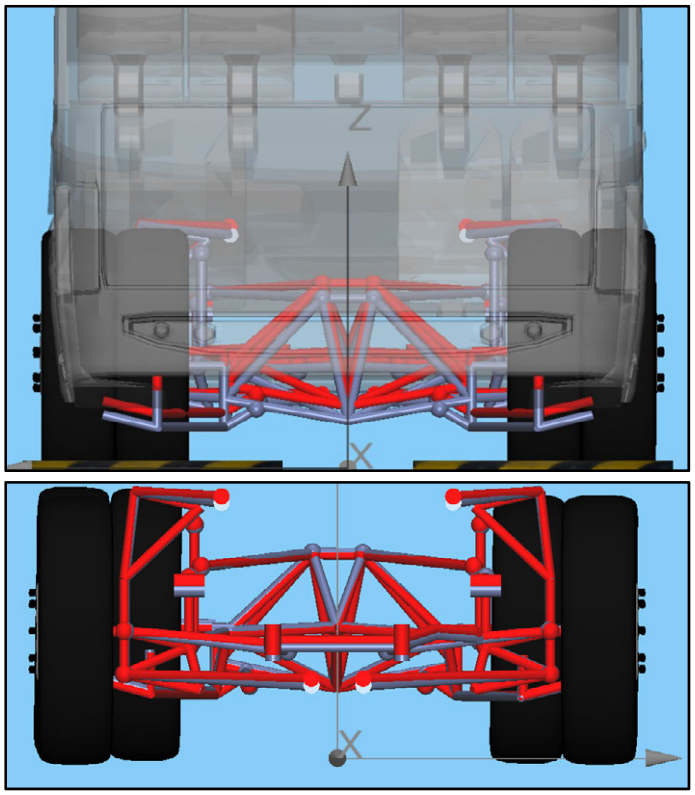


Figure 6.17: 3D comparison between original and optimized suspensions (comfort). Top: speed bumps test. Bottom: four-post test.

6.5 Multi-objective optimization

When handling and ride comfort responses are tackled separately, unrealistic solutions are found, at least for mainstream vehicles, where a minimum level of both handling and comfort have to be guaranteed. In this subsection, both objective functions are optimized using the Pareto-optimal multi-objective approach presented at the beginning of this chapter.

In order to visualize the basic concepts, a simplified problem with only two parameters (#1 and #2 in Table 6.1) and both handling and comfort objective functions is solved first. By defining a uniformly-distributed (*exhaustive*) sequence of design parameters, the design space is plotted in a 2D chart (see Figure 6.18(a)). The corresponding objective function space is then computed and plotted (see Figure 6.18(b)). In these figures, blue dots correspond to parameter configurations that fulfill optimization constraints (and thus are feasible points), and red dots to infeasible configurations. This way of computing the objective functions is very inefficient, but shows the Pareto front effectively, and helps to assess the optimality of the different parameter configurations.

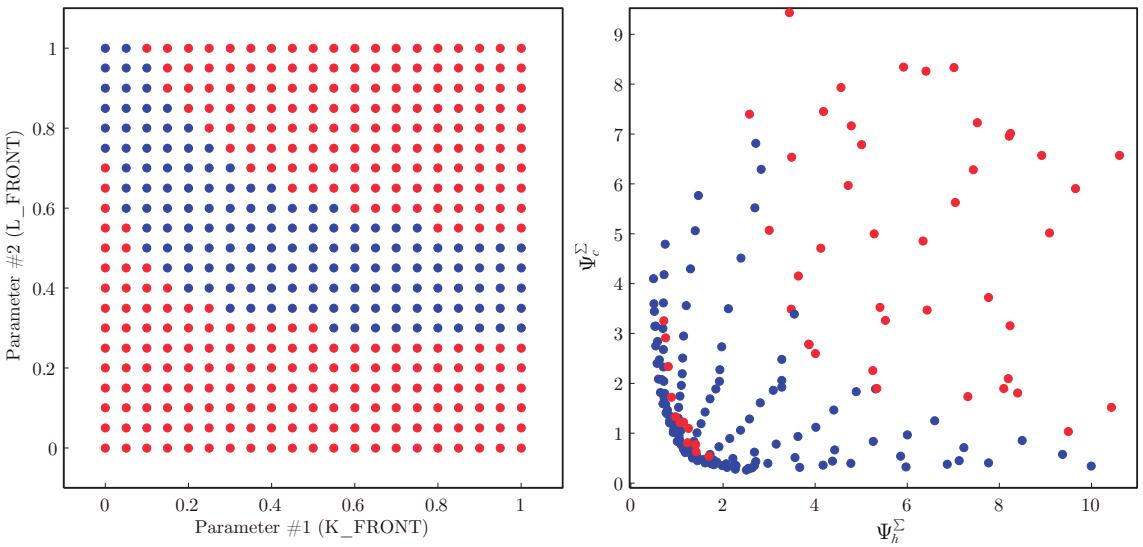


Figure 6.18: Design and objective function spaces.

In the real multi-objective problem, however, the full set of design parameters is considered, together with the aforementioned objective functions. Regarding the optimization algorithms, MATLAB's `fminimax` function has been employed. The reason why no alternative global methods have been used is that the available tools did not support nonlinear constraints.

The results of the real multi-objective optimization problem are shown in Figure 6.19, where the objective function values during the optimization process are plotted. The history of design parameter values can be seen in Figure 6.20, while the final absolute values are gathered in Table 6.6. In this case, the reduction of the objective functions is obviously smaller than when handling and com-

fort were optimized independently, because now a trade-off between them has to be achieved. Nevertheless, the objective function reductions are, respectively, 71% and 70%, which represent a considerable improvement of handling and ride comfort behaviors.

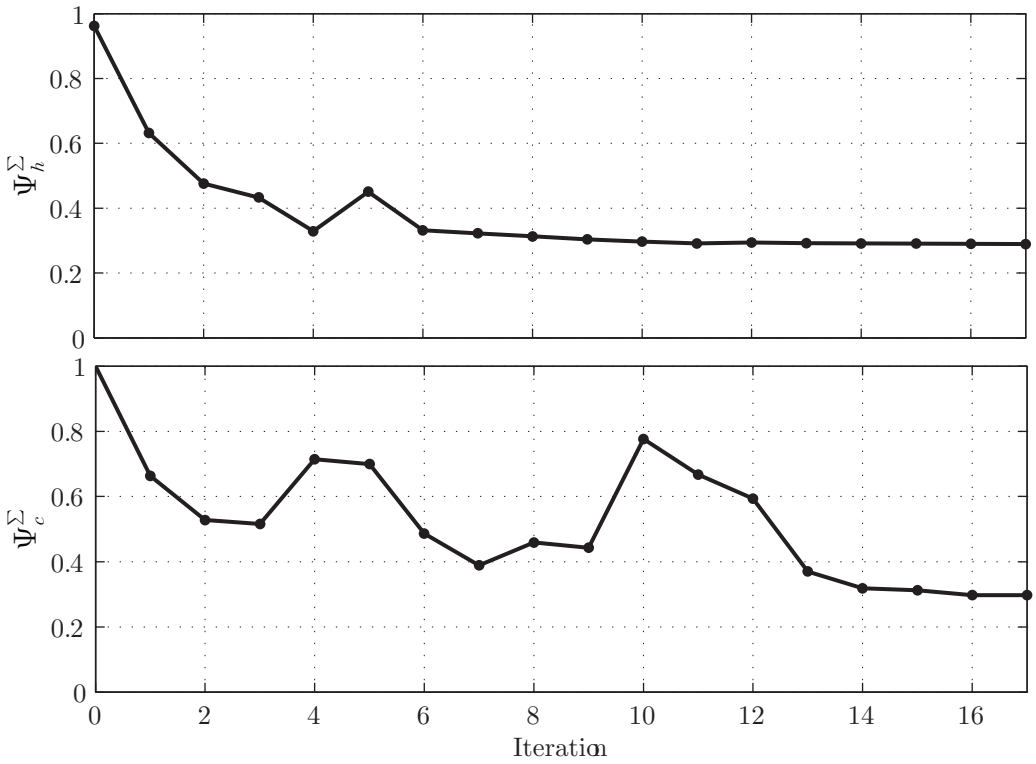


Figure 6.19: Objective function values during the multi-objective optimization.

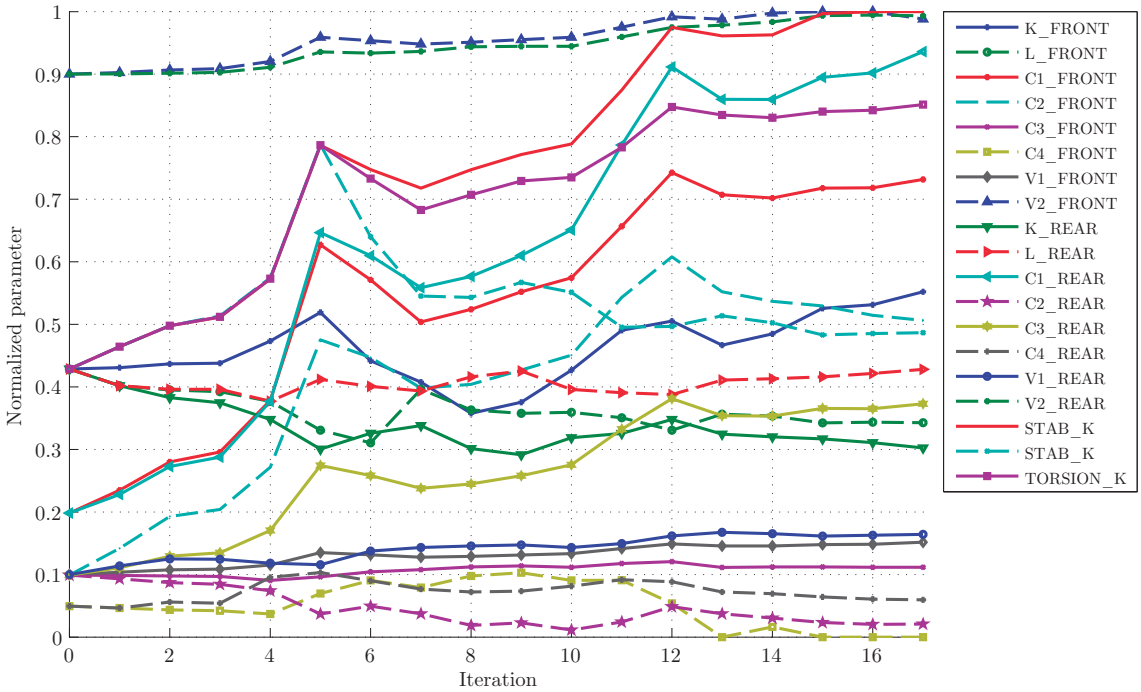


Figure 6.20: Design parameter values during the multi-objective optimization.

Objective or parameter	Initial value	Final value
Ψ_h^{Σ}	$1.00\times10^{+0}$	2.89×10^{-1}
Ψ_c^{Σ}	$1.00\times10^{+0}$	2.97×10^{-1}
1	$5.56\times10^{+5}$	$6.56\times10^{+5}$
2	2.35×10^{-1}	2.00×10^{-1}
3	$1.98\times10^{+4}$	$7.18\times10^{+4}$
4	$9.92\times10^{+3}$	$5.14\times10^{+4}$
5	$9.92\times10^{+3}$	$1.12\times10^{+4}$
6	$4.96\times10^{+3}$	1.20×10^{-12}
7	1.00×10^{-2}	1.48×10^{-2}
8	-1.00×10^{-2}	$0.00\times10^{+0}$
9	$1.40\times10^{+5}$	$1.11\times10^{+5}$
10	3.08×10^{-1}	3.04×10^{-1}
11	$1.98\times10^{+4}$	$9.02\times10^{+4}$
12	$9.92\times10^{+3}$	$2.02\times10^{+3}$
13	$9.92\times10^{+3}$	$3.65\times10^{+4}$
14	$4.96\times10^{+3}$	$6.06\times10^{+3}$
15	1.00×10^{-2}	1.63×10^{-2}
16	-1.00×10^{-2}	-5.52×10^{-4}
17	$2.16\times10^{+5}$	$4.32\times10^{+5}$
18	$2.00\times10^{+5}$	$2.20\times10^{+5}$
19	$3.00\times10^{+5}$	$5.17\times10^{+5}$

Table 6.6: Final objective function and parameter values (multi-objective).

6.6 Implementation and efficiency

Running and controlling the optimization process from MATLAB certainly has interesting advantages. Most importantly: the availability of out-of-the-box state-of-the-art optimization algorithms; easy-to-use postprocessing and plotting functions; and a straightforward implementation. However, it is key for good computational efficiency that the evaluation of the objective functions is carried out efficiently. For this reason, the multibody simulation itself is written in C/C++ code and encapsulated in a MEX-function, which can then be called from the optimization script in MATLAB.

One could wonder, however, how much of the computational load of an entire optimization process can be attributed to the interpretation of MATLAB code, and how much to the C/C++ code. Table 6.7 shows the CPU time profile of a typical optimization procedure. The total computation time is 2885.189 s. Functions are sorted by self CPU time, that is, the time spent by the program inside

the function, without counting the time spent in the children functions. The number of function calls and the total time (counting the children function times) are also displayed. It is obvious that the MEX-function, which is efficiently programmed in C/C++ code, takes most of the elapsed time (around 97%). Therefore, the overhead caused by MATLAB is fairly small, and does not affect the global efficiency of the optimization process.

Function name	# Calls	Total time (s)	Self time (s)
`EXP_coach_MEX` (MEX-file)	598	2827.977	2827.977
`ghostscript`	19	23.968	21.235
`mbs3d`	1	2884.577	10.479
`ghostscript>check_gs_path`	19	2.638	2.638
`graphics\private\prepareui`	19	1.757	1.728
`graphics\private\loadfonts`	1	1.251	1.251
`checkIntegrationOpt`	1	7.896	1.124
`cross`	28046	1.381	1.103
`isprop`	13222	0.918	0.918
`findobjhelper`	1047	0.901	0.901
`export_fig`	19	33.146	0.588
`hardcopy`	19	1.527	0.521

Table 6.7: Profile of CPU times in a typical optimization procedure in MATLAB.

Finally, a brief comparison of optimization methods is carried out. To that end, a single handling objective function, Ψ_{BMP1} in Eq. (6.42), is considered. Six optimization methods with standard properties have been tested, among which there are gradient-based methods (SQP and interior-point), penalty methods (quasi-Newton and trust-region), and global methods (GA and hybrid).

Method	O. F. reduction (%)	O. F. evaluations	Elapsed time (h)
`sqp`	32.73	579	1.73
`interior-point`	30.79	881	2.28
`quasi-newton`	28.99	1638	4.88
`trust-region`	28.99	1638	5.03
`genetic-algorithm`	21.89	2250	3.16
`hybrid`	26.09	4736	6.55

Table 6.8: Experiments with other optimization methods.

SQP is almost always faster and more effective than the other algorithms, although the interior-point method is also competitive. Penalty methods take longer computation times, they are more sensible to changes in the penalty factors, and incur infeasible configurations more easily. Global methods have to be run for only a few generations to be competitive, which in turn implies a smaller objective function reduction.

Chapter 7

Conclusions

During the last two decades, several authors in the field of multibody systems optimization and design have raised a challenging question: Is it possible to develop a fairly general-purpose, accurate, efficient software for the sensitivity analysis of real-life mechanical systems? From the author's point of view, the answer is yes. Such a program would require efficient and scalable methods for the algorithmic differentiation of dynamic expressions. In the medium term, automatic differentiation tools are envisaged as the best candidates for that task, as it has been thoroughly shown in this Thesis. From the very definition of the multibody system model, through the algorithmic computation of its design sensitivities, to the dynamic response optimization, a somewhat general-purpose approach for the optimization of mechanical systems has been presented herein.

7.1 Contributions

The optimization problem tackled in this Thesis has required the analysis and implementation of a wide spectrum of numerical problems, from the dynamic simulation of multibody systems to the efficient nonlinear programming with nonlinear constraints, including automatic differentiation techniques and complex sensitivity analyses. From the beginning, a trade-off between theoretical originality and the analysis of real-life example systems has been sought, aiming at the solution of the global response optimization problem.

The main conclusions drawn from the work developed in this Thesis can be summarized as follows:

- The dynamic response optimization of multibody systems is still a very challenging research topic, given the complexity and variety of numerical problems involved (forward multibody dynamics, sensitivity analysis, numerical optimization, etc.), the associated efficiency issues, and the difficulty of interpreting the results. This might explain the delay in the implementation of optimization techniques on multibody software packages, in contrast with other areas like the finite element method, where robust and general-purpose optimization tools are well established.
- A particularly efficient formulation for the dynamic simulation of multibody systems, namely the double-step Maggi's formulation, has been implemented, and the recursive computation of joint reactions has been developed. Based on this formulation, an 18-DOF coach vehicle has been modeled in detail with the real suspension geometry, and including nonlinear dampers, accurate tire forces, approximated linear chassis flexibility, a traction controller and realistic steering inputs. The computation times of the coach simulations are shorter (around 37% for a typical case) than the ones of state-of-the-art commercial multibody packages.
- A benchmarking of two automatic differentiation tools in the field of multibody dynamics has been carried out, using an implicit integrator as a sample problem. State-of-the-art automatic differentiation tools ADOL-C and ADIC2 have been successfully implemented for the computation of the key Jacobian matrix, the latter being investigated in close collaboration with the developers themselves. Computational times are close to those of numerical differentiation, and in some large cases are shorter (up to a 60% time reduction), especially in cases where the Jacobian matrix is sparse. The time required to implement ADOL-C has been found to be shorter than the one of ADIC2, in addition to being more general, and the system sparsity can be exploited algorithmically without having to modify the original code.
- Then, a hybrid direct-automatic differentiation method has been developed for the computation of independent state sensitivities, based on direct-differentiation sensitivity equations and on the systematic use of an automatic-differentiation tool, ADOL-C. This approach allows computing machine-precision state sensitivities and the gradient of any objective function with respect to any design parameter in a completely automated way, by propagating the derivatives through the code. Several implementation approaches have been studied. The efficiency of automatic differentiation, in this case, is still behind the one of numerical differentiation. Nevertheless, more accurate parameter relevance analyses can be performed, which come in handy when the designer wants to perform response optimization.

– The formulation for the computation of sensitivities has been applied to the dynamic response optimization of the coach. Specifically, handling and ride comfort objectives have been considered, each of them consisting of several real-life maneuvers. These objectives have been considered both individually and combined into a Pareto-optimal multi-objective approach, as a realistic design optimization of vehicles demands. Local and global optimization methods have been tested, and improvements of up to 71% have been achieved in the multi-objective problem.

Overall, a somewhat complete theoretical and practical analysis for the dynamic response optimization of multibody systems has been presented, from the equations of motion to the dynamic response optimization, including a real vehicle with real constraints and under real maneuvers. Performance has been explored at all stages of the development by running more than eight numerical examples with various topologies, configurations and time-steps, and useful efficiency conclusions have been drawn.

7.2 Future work

Automatic differentiation tools, and specifically operator overloading tools like ADOL-C, have reached such a level of maturity that certain industrial applications of mechanical optimization are no longer a dream. For example, ADOL-C could be implemented in commercial multibody packages like Adams following the ideas presented here, with a reasonable effort. This would provide accurate, general-purpose sensitivities of mechanical systems (even containing user-defined functions and scripts), which would eventually enable gradient-based optimization. Next, a few suggestions for future development are presented concisely.

– The efficiency of the presented hybrid direct-automatic differentiation approach for the computation of sensitivities should be put to the test with larger and more complex multibody systems. Specifically, a deep analysis of how the size and topology of the multibody system affect the efficiency of automatic differentiation remains to be carried out. In real multibody systems like the coach vehicle, the algorithm should be adapted to flexible multibody formulations to capture finite deformations. This would certainly involve adapting the methods to larger problem sizes.

- As suggested by algorithmic differentiation developers themselves, it would be worth trying the automatic differentiation (with respect to the parameters) of the whole sensitivity integration process (rather than just the differentiation of the state vector derivative), for the computation of state sensitivities. As explained, this strategy would have to make use of state-of-the-art checkpointing techniques, in order to reduce the computational burden associated to a huge computational tree. From the point of view of user implementation, this would be the most important alternative towards the improvement of the sensitivity analysis efficiency.
- Regarding the multibody formalism, the implicit integration of motion and sensitivity equations can be further explored, as well as the implementation of variable time-step methods. The former might be interesting because a small number of iterations (and usually operations) benefits automatic differentiation tools, and the latter would take advantage of the simplicity with which the direct differentiation method handles variable time-steps, in contrast with the adjoint variable method. Also, since the reverse mode of automatic differentiation is the (discrete) numerical equivalent of the (continuous) adjoint variable method, it would be interesting to compare both approaches. These improvements would facilitate the eventual development of real-time sensitivity analyses, which, to the author's knowledge, are still far from feasible. Also, the introduction of ad-hoc parallelization techniques seems to be a necessary step in this direction.
- Only a glimpse of the potential of automatically differentiated quantities has been presented in this Thesis, namely the dynamic response optimization of multibody systems. Neighbor areas like geometrical optimization and test-based parameter identification could also benefit from the strategies presented here. Very little work has addressed the algorithmic computation of derivatives in these fields.
- Finally, a step can be taken towards a deeper exploitation of sensitivity information in the field of vehicle dynamics. This can be interesting not only for the computation of gradients, but also, for example, for the implementation of on-board sensitivity-analysis-based controllers. Moreover, sensitivity data can be used to improve the selection of objective functions and maneuvers, in order to capture the significant behavior more accurately and not just through trial-and-error procedures.

References

Ambrósio, J.A.C. and Gonçalves, J. (1999) Complex flexible multibody systems with application to vehicle dynamics, in Ambrósio, J.A.C. and Schiehlen, W.O. *Advances in Computational Multibody Dynamics*, IDMEC/IST.

Ambrósio, J.A.C. and Verissimo, P. (2009) Improved bushing models for general multibody systems and vehicle dynamics, *Multibody System Dynamics*, vol. 22, pp. 341–365.

Amirouche, F.M.L. (1992) *Computational Methods for Multibody Dynamics*, Prentice-Hall.

Anderson, K.S. and Hsu, Y. (2002) Analytical fully-recursive sensitivity analysis for multibody dynamic chain systems, *Multibody System Dynamics*, vol. 8, pp. 1–27.

Andersson, D. and Eriksson, P. (2004) Handling and ride comfort optimisation of an intercity bus, *Vehicle System Dynamics Supplement*, vol. 41, pp. 547–556.

Arora, J.S. (1989) *Introduction to Optimum Design*, Singapore: McGraw-Hill.

Ashrafiuon, H. and Mani, N.K. (1990) Analysis and optimal design of spatial mechanical systems, *ASME Journal of Mechanical Design*, vol. 112, pp. 200–207.

Avilés, R., Ajuria, M.B. and García de Jalón, J. (1985) A Fairly General Method for the Optimum Synthesis ol Planar Mechanisms, *Mechanisms and Machine Theory*, vol. 20, pp. 321–328.

Banerjee, J.M. and McPhee, J. (2013) Symbolic Sensitivity Analysis of Multibody Systems, in Samin, J.-C. and Fisette, P. (ed.) *Multibody Dynamics*.

Barthelemy, J.-F. and Hall, L.E. (1992) *Automatic differentiation as a tool in engineering design*.

Bauchau, O. and Laulusa, A. (2008) Review of Contemporary Approaches for Constraint Enforcement in Multibody Systems, *Journal of Computational and Nonlinear Dynamics*, vol. 3, pp. 011005:1–8.

Baumal, A.E., McPhee, J.J. and Calamai, P.H. (1998) Application of genetic algorithms to the design optimization of an active vehicle suspension system, *Comput. Methods Appl. Mech. Engrg.*, vol. 163, pp. 87–94.

Baumgarte, J. (1972) Stabilization of Constraints and Integrals of Motion in Dynamical Systems, *Computer Methods in Applied Mech. and Engineering*, pp. 1–16.

Bayo, E. and Ledesma, R. (1996) Augmented Lagrangian and Mass-Orthogonal Projection Method for Constrained Multibody Dynamics, *Journal of Nonlinear Dynamics*, vol. 9, pp. 113–130.

Bendsoe, M.P. and Sigmund, O. (2003) *Topology Optimization: Theory, Methods and Applications*, Springer.

Berz, M., Bischof, C., Corliss, G. and Griewank, A. (1996) *Computational Differentiation: Techniques, Applications, and Tools*, Philadelphia: SIAM.

Besselink, I. and Van Asperen, F. (1994) Numerical Optimization of the Linear Dynamic Behaviour of Commercial Vehicles, *Vehicle System Dynamics*, vol. 23, pp. 53–70.

Bestle, D. (1994) *Analyse und Optimierung von Mehrkörpersystemen*, Springer-Verlag.

Bestle, D. and Eberhard, P. (1992) Analyzing and Optimizing Multibody Systems, *Mech. Struct. & Mach.*, vol. 20, no. 1, pp. 67–92.

Bischof, C.H. (1996) On the automatic differentiation of computer programs and an application to multibody systems, IUTAM Symposium on Optimization of Mechanical Systems.

Bischof, C., Carle, A., Hovland, P., Khademi, P. and Mauer, A. (1995) *Adifor 2.0 User's Guide - Revision D.*

Bischof, C.H., Hovland, P.D. and Norris, B. (2008) On the implementation of automatic differentiation tools, *Higher-Order Symb Comput*, vol. 21, pp. 311–331.

Bischof, C., Khademi, P., Mauer, A. and Carle, A. (1996) Adifor 2.0: Automatic differentiation of Fortran 77 programs, *Numerical Algorithms Symbolic Computing, IEEE Computational Science & Engineering.*

Bischof, C.H., Roh, L. and Mauer, A. (1997) ADIC — An extensible automatic differentiation tool for ANSI-C, *Software–Practice and Experience*, vol. 27, pp. 1427–1456.

Bras, L.F.D. and Azevedo, A.F.M. (2001) Object oriented implementation of a second-order optimization method, *Computer Aided Optimum Design of Structures VII.*

Brüls, O. and Eberhard, P. (2008) Sensitivity analysis for dynamic mechanical systems with finite rotations, *Int. J. Numer. Meth. Engng*, vol. 74, pp. 1897–1927.

Callejo, A. and García de Jalón, J. (2011) Automatic differentiation of forces in forward multibody dynamics, ECCOMAS Multibody Dynamics 2011.

Ceccarelli, M. (1993) Proceedings of the Optimal Design and Location of Manipulators NATO - Advanced Study Institute on Computer Aided Analysis of Rigid and Flexible Mechanical Systems. Volume II, 299–310.

Chang, C.O. and Nikravesh, P.E. (1985) Optimal design of mechanical systems with constraint violation stabilization method, *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 107, pp. 493–498.

Chatillon, M.M., Jezequel, L., Coutant, P. and Baggio, P. (2006) Hierarchical optimisation of the design parameters of a vehicle suspension system, *Vehicle System Dynamics*, pp. 817–839.

Chen, K. and Beale, D.G. (2003) Base Dynamic Parameter Estimation of a McPherson Suspension Mechanism, *Vehicle System Dynamics*, vol. 39, pp. 227–244.

Collard, J.-F. (2007) *Geometrical and Kinematic Optimization of Closed-Loop Multibody Systems* (Doctoral Thesis), Université Catholique de Louvain.

Cuadrado, J., Dopico, D., González, M. and Naya, M.A. (2004) Naya, A combined penalty and recursive real-time formulation for multibody dynamics, *Journal of Mechanical Design*, vol. 126, pp. 602–608.

Curtis, A.R., Powell, M.J.D. and Reid, J.K. (1974) On the Estimation of Sparse Jacobian Matrices, *J. Inst. Maths Applics*, vol. 13, pp. 117–119.

De-Juan, A., Sancibrian, R. and Viadero, F. (2012) Optimal Synthesis of function generation in steering linkages, *International Journal of Automotive Technology*, vol. 13, no. 7, p. 1033—1046.

Dias, J.M.P. and Pereira, M.S. (1997) Sensitivity Analysis of Rigid-Flexible Multibody Systems, *Multibody System Dynamics*, vol. 1, pp. 303–322.

Ding, J.-Y., Pan, Z.-K. and Chen, L.-Q. (2007) Second order adjoint sensitivity analysis of multibody systems described by differential–algebraic equations, *Multibody System Dynamics*, vol. 18, pp. 599–617.

Dopico, D. (2004) *Formulaciones semi-recursivas y de penalización para la dinámica en tiempo real de sistemas multicuerpo* (Doctoral Thesis), Universida-de da Coruña.

Dürrbaum, A., Klier, W. and Hahn, H. (2002) Comparison of automatic and symbolic differentiation in mathematical modeling and computer simulation of rigid-body systems, vol. 7, pp. 331–355.

Eberhard, P. (1996) Adjoint Variable Method for the Sensitivity Analysis of Multibody Systems Interpreted as Continuous, Hybrid Form of Automatic Differentiation SIAM.

Eberhard, P. and Bischof, C. (1999) Automatic differentiation of numerical integration algorithms, *Mathematics of Computation*, vol. 68, no. 226, pp. 717–731.

Eberhard, P. and Schiehlen, W. (2006) Computational dynamics of multibody systems: History, formalisms, and applications, *Journal Computational and Nonlinear Dynamics*, vol. 1, pp. 3–12.

Eberhard, P., Schiehlen, W. and Bestle, D. (1999) Some advantages of stochastic methods in multicriteria optimization of multibody systems, *Archive of Applied Mechanics*, vol. 69, pp. 543–554.

Eberhard, P., Schiehlen, W. and Sierts, J. (2007) Sensitivity analysis of inertia parameters in multibody dynamics simulations, 12th IFToMM World Congress, Besançon.

Enciu, P., Gerbaud, L. and Wurtz, F. (2010) Automatic Differentiation Applied for Optimization of Dynamical Systems, *IEEE Transaction on magnetics*, vol. 46, no. 8, pp. 2943–2946.

Erdman, A.G. (1995) Computer-Aided Mechanism Design: Now and the future, *ASME 50th Anniversary of Design Engineering Division, Special Combined Issue of Journal of Mechanical Design and Journal of Vibration and Acoustics*, vol. 117, no. B, pp. 93–100.

Eriksson, P. and Friberg, O. (2000) Ride comfort optimization of a city bus, *Struct Multidisc Optim*, vol. 20, pp. 67–75.

Etman, L.F.P. (1997) *Optimization of multibody systems using approximation concepts* (Doctoral Thesis), Technische Universiteit Eindhoven.

Etman, L.F.P., Van Campen, D.H. and Schoofs, A.J.G. (1998) Design Optimization of Multibody Systems by Sequential Approximation, *Multibody System Dynamics*, vol. 2, pp. 393–415.

Featherstone, R. (1983) The Calculation of Robot Dynamics Using Articulated-Body Inertias, *International Journal of Robotics Research*, vol. 2, pp. 13–30.

Fisette, P., Raucent, B. and Samin, J.C. (1996) Minimal Dynamic Characterization of Tree-Like Multibody Systems, *Nonlinear Dynamics*, vol. 9, pp. 165–184.

Fletcher, R. (1987) *Practical Methods of Optimization*, 2nd edition, John Wiley & Sons.

García de Jalón, J. (2007) Twenty-five years of natural coordinates, *Multibody System Dynamics*, vol. 18, pp. 15–33.

García de Jalón, J., Álvarez, E., De Ribera, F.A., Rodríguez, I. and Funes, F.J. (2005) A fast and simple semi-recursive dynamic formulation for multi-rigid-body systems, in Ambrósio, J. *Advances in Computational Multibody Systems*, Springer-Verlag.

García de Jalón, J. and Bayo, E. (1994) *Kinematic and Dynamic Simulation of Multi-Body Systems – The Real-Time Challenge*, New York: Springer-Verlag.

García de Jalón, J., Callejo, A. and Hidalgo, A.F. (2012) Efficient Solution of Maggi's Equations, *Journal of Computational and Nonlinear Dynamics*, vol. 7, pp. 021003:1–10.

García de Jalón, J. and Gutiérrez-López, M.D. (2013) Multibody dynamics with redundant constraints and singular mass matrix: existence, uniqueness, and determination of solutions for accelerations and constraint forces, *Multibody System Dynamics*, pp. 1–31.

García de Jalón, J., Hidalgo, A.F. and Callejo, A. (2009) MBS software development with Matlab for teaching and industrial usages, Proceedings of the ASME IDETC/CIE Conference, San Diego.

García de Jalón, J., Unda, J., Avello, A. and Jiménez, J.M. (1987) Dynamic Analysis of Three-Dimensional Mechanisms in Natural Coordinates, *ASME Journal of Mechanisms, Transmissions and Automation in Design*, vol. 109, pp. 460–465.

Gauthier, M. and Khalil, W. (1990) Direct Calculation of Minimum Set of Inertial Parameters of Serial Robots, *IEEE Transactions on Robotics and Automation*, vol. 6, pp. 368–373.

Gebremedhin, A.H., Nguyen, D., Patwary, M. and Pothen, A. (2011) *ColPack: Software for Graph Coloring and Related Problems in Scientific Computing*, Purdue University.

Georgiou, G., Badarlis, A. and Natsiavas, S. (2008) Modelling and ride dynamics of a flexible multi-body model of an urban bus, IMechE Part K: J. Multi-body Dynamics, 143–154.

Gombor, B. (2005) Dynamic analysis of a bus body frame: determination of the loads and stresses, *Vehicle System Dynamics*, vol. 43, no. 11, pp. 807–822.

Gómez-Cristóbal, J.A. (2003) *Método de síntesis dimensional óptima de sistemas multicuerpo con restricciones dinámicas. Aplicación al diseño de mecanismos planos* (Doctoral Thesis), Universidad de La Rioja.

Gonçalves, J.P.C. (2002) *Rigid and flexible multibody systems optimization for vehicle dynamics* (Doctoral Thesis), Instituto Superior Técnico.

Gonçalves, J.P.C. and Ambrósio, A.C. (2005) Road Vehicle Modeling Requirements for Optimization of Ride and Handling, *Multibody System Dynamics*, vol. 13, pp. 3–23.

Griewank, A. (1989) On automatic differentiation, in Iri, M. and Tanabe, K. *Mathematical Programming: Recent Developmentes and Applications*, Dordech: Kluwer Academic Publishers.

Griewank, A. (1993) Some bound on the complexity of gradients, jacobians and Hessians, in Pardalos, P.M. *Complexity in Nonlinear Optimization*, World Scientific Publishers Co.

Griewank, A., Bischof, C.H., Corliss, G.F., Carle, A. and Williamson, K. (1993) Derivative convergence for iterative equation solvers, *Optimization Methods and Software*, vol. 2, pp. 321–355.

Griewank, A., Juedes, D. and Utke, J. (1996) ADOL–C, a package for the automatic differentiation of algorithms written in C/C++, *ACM Trans. Math. Software*, vol. 22, pp. 131–167.

Griewank, A. and Reese, S. (1992) *On the Calculation of Jacobian Matrices by the Markowitz Rule*, Argonne National Laboratory.

Griewank, A. and Walther, A. (2008) *Evaluating derivatives - Principles and Techniques of Algorithmic Differentiation*, 2nd edition, SIAM.

Griffin, M.J. (2007) Discomfort from feeling vehicle vibration, *Vehicle System Dynamics*, vol. 45, no. 7–8, pp. 679–698.

Gutiérrez-López, M.D., Callejo, A. and García de Jalón, J. (2012) Computation of independent sensitivities using Maggi's formulation, 2nd Joint International Conference on Multibody System Dynamics, Stuttgart.

Haghiac, H., Haque, I. and Fadel, G. (2004) An assessment of a genetic algorithm-based approach for optimising multi-body systems with applications to vehicle handling performance, *Int. J. Vehicle Design*, vol. 36, no. 4, pp. 320–344.

Hannemann, R., Marquardt, W., Naumann, U. and Gendler, B. (2010) Discrete first and second-order adjoints and automatic differentiation for the sensitivity analysis of dynamic models, *Procedia Computer Science - ICCS 2010*, no. 1, pp. 297–305.

Haug, E.J. (1989) *Computer-Aided Kinematics and Dynamics of Mechanical Systems, Volume I: Basic Methods*, Allyn and Bacon.

Haug, E.J. and Arora, J.S. (1978) Design Sensitivity Analysis of Elastic Mechanical Systems, *Computer Methods in Applied Mechanics and Engineering*, vol. 15, pp. 35–62.

Haug, E.J., Choi, K.K. and Komkov, V. (1986) *Design Sensitivity Analysis of Structural Systems*, New York: Academic Press.

Haug, E.J., Wehage, R.A. and Mani, N.K. (1984) Design sensitivity analysis of large-scale constrained dynamic mechanical systems, *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 106, pp. 156–162.

Hegazy, S., Rahnejat, H. and Hussain, K. (2000) Multi-Body Dynamics in Full-Vehicle Handling Analysis under Transient Manoeuvre, *Vehicle System Dyn.*, vol. 34, pp. 1–24.

Hidalgo, A.F. (2013) *Simulación en tiempo real de vehículos industriales con modelos multicuerpo de gran complejidad* (Doctoral Thesis), Universidad Politécnica de Madrid.

Holland, J.H. (1962) Outline of a logical theory of adaptive systems, *Journal of ACM*, vol. 3, pp. 297–314.

Huston, R.L. (1990) *Multibody Dynamics*, Butterworth-Heinemann.

Jazar, R.N. (2008) *Vehicle Dynamics: Theory and Application*, Springer.

Jerkovsky, W. (1978) The Structure of Multibody Dynamic Equations, *Journal of Guidance and Control*, vol. 1, pp. 173–182.

Juedes, D.W. (1991) *A taxonomy of automatic differentiation tools*, Department of Computer Science - Iowa State University.

Kalman, R.E. (1960) A New Approach to Linear Filtering and Prediction Problems, *Transaction of the ASME - Journal of Basic Engineering*, vol. 3, pp. 35–45.

Kanarachos, S. (2012) A new min-max methodology for computing optimised obstacle avoidance steering manoeuvres of ground vehicles, *Int. Journal of Systems Science*.

Kawamoto, A. (2004) *Generation of articulated mechanisms by optimization techniques* (Doctoral Thesis), Technical University of Denmark.

Kawamoto, A. (2005) Path-generation of articulated mechanisms by shape and topology variations in non-linear truss representation, *International Journal for Numerical Methods in Engineering*, vol. 64, pp. 1557–1574.

Kelley, C.T. (1999) *Iterative Methods for Optimization*, SIAM - Frontiers in Applied Mechanics.

Kim, S.-S. (2002) A Subsystem Synthesis Method for Efficient Vehicle Multibody Dynamics, *Multibody System Dynamics*, vol. 7, pp. 189–207.

Kim, M.-S. and Choi, D.-H. (1998) Min-max dynamic response optimization of mechanical systems using approximate augmented Lagrangian, *International Journal for Numerical Methods in Engineering*, vol. 43, pp. 549–564.

Kim, S.-S. and Vanderploeg, M.J. (1986) A General and Efficient Method for Dynamic Analysis of Mechanical Systems Using Velocity Transformations, *ASME Journal of Mechanisms, Transmissions and Automation in Design*, vol. 108, pp. 176–182.

Kirkpatrick, S., Gelatt, G.C. and Vecchi, M.P. (1983) Optimization by Simulated Annealing, *Science*, vol. 220, no. 4598.

Krishnaswami, P. and Bhatti, M.A. (1984) A General Approach for Design Sensitivity Analysis of Constrained Dynamic Systems, *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, pp. 84-DET-132.

Kübler, L., Henninger, C. and Eberhard, P. (2005) Multi-Criteria Optimization of a Hexapod Machine, *Multibody System Dynamics*, vol. 14, pp. 225–250.

Laulusa, A. and Bauchau, O.A. (2008) Review of Classical Approaches for Constraint Enforcement in Multibody Systems, *Journal of Computational and Nonlinear Dynamics*, vol. 3, no. 1, p. 011004.

Lechner, D., Ferrandez, F. and Fleury, D. (1983) *Manoeuvres et sollicitations en situation de urgente*, France: INRETS.

Li, S., Petzold, L. and Zhu, W. (2000) Sensitivity analysis of differential–algebraic equations: A comparison of methods on a special problem, *Applied Numerical Mathematics*, vol. 32, pp. 161–174.

Liu, X. (1996) Sensitivity analysis of constrained flexible multibody systems with stability considerations, *Mech. Math. Theory*, vol. 31, no. 7, pp. 859–863.

Luh, J.Y.S., Walker, M.W. and Paul, R.P.C. (1980) On Line Computational Scheme for Mechanical Manipulators, *ASME Journal of Dynamic Systems, Measurements and Control*, vol. 102, pp. 69–76.

Luque, P. and Mántaras, D.A. (2003) Pneumatic suspensions in semi-trailers: Part I – General considerations and simplified models, *Int. J. of Heavy Vehicle Systems*, vol. 10, p. 4.

Luque, P. and Mántaras, D.A. (2003) Pneumatic suspensions in semi-trailers: Part II – Computer simulation, *Int. J. of Heavy Vehicle Systems*, vol. 10, p. 4.

Maly, T. and Petzold, L.R. (1996) Numerical methods and software for sensitivity analysis of differential-algebraic systems, *Applied Numerical Mathematics*, vol. 20, pp. 57–79.

Martín, A.L. (2013) *Estudio teórico experimental de la estabilidad lateral en vehículos cisterna. Metodología para la determinación del umbral de vuelco* (Doctoral Thesis), Madrid: Universidad Politécnica de Madrid.

Martz, P. (2007) *OpenSceneGraph quick start guide — A quick introduction to the cross-platform Open source Scene Graph API*, Skew Matrix Software LLC.

Mastinu, G., Gobbi, M. and Miano, C. (2006) *Optimal Design of Complex Mechanical Systems*, New York: Springer.

McGarva, J.R. (1994) Rapid Search and Selection of Path Generating Mechanisms from a Library, *Mechanism and Machine Theory*, vol. 29, no. 2, pp. 223–235.

Moore, B., Kövecses, J. and Piedboeuf, J.C. (2003) Symbolic Model Formulation for Dynamic Parameter Identification, Multibody Dynamics IDME/IST, Lisbon.

Morandini, M. (2006) *Integration of automatic differentiation tools within object-oriented codes: accuracy and timings - Deliverable 2.2*, Dipartimento di Ingegneria Aerospaziale, Politecnico di Milano.

Narayanan, S.H.K., Norris, B., Hovland, P., Nguyen, D.C. and Gebremedhin, A.H. (2011) Sparse jacobian computation using ADIC2 and ColPack, *Procedia Computer Science - ICCS 2011*, vol. 4, pp. 2115–2123.

Narayanan, S.H.K., Norris, B. and Winnicka, B. (2010) ADIC2: Development of a component source transformation system for differentiating C and C++, *Procedia Computer Science - ICCS 2010*, pp. 1845–1853.

Naudé, A.F. and Snyman, J.A. (2003) Optimisation of road vehicle passive suspension systems. Part 1. Optimisation algorithm and vehicle model, *Applied Mathematical Modelling*, vol. 27, pp. 249–261.

Naudé, A.F. and Snyman, J.A. (2003) Optimisation of road vehicle passive suspension systems. Part 2. Qualification and case study, *Applied Mathematical Modelling*, vol. 27, pp. 263–274.

Naumann, U. and Walther, A. (2003) *An introduction to using software tools for automatic differentiation*, Mathematics and Computer Science Division - Argonne National Laboratory.

Negrut, D., Serban, R. and Potra, F.A. (1997) A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics. Joint Formulation, *Mechanics of Structures and Machines*, vol. 25, no. 2.

Nikravesh, P.E. (1988) *Computer-Aided Analysis of Mechanical Systems*, Prentice-Hall.

Nocedal, J. and Wright, S.J. (2006) *Numerical Optimization*, Springer.

Orlandea, N., Chace, M.A. and Calahan, D.A. (1977) A Sparsity Oriented Approach to the Dynamic Analysis and Design of Mechanical Systems, *ASME Journal of Engineering for Industry*, vol. 99, pp. 773–784.

Özcan, D., Sönmez, Ü. and Güvenç, L. (2013) Optimisation of the Nonlinear Suspension Characteristics of a Light Commercial Vehicle, *International Journal of Vehicular Technology*, vol. 2013, no. 562424.

Pacejka, H.B. (1996) The tyre as a vehicle component, XXVI Fisita Congress, Prague.

Pacejka, H.B. (2002) *Tyre and vehicle dynamics*, Elsevier.

Paeng, J.K. and Arora, J.S. (1989) Dynamic Response Optimization of Mechanical Systems With Multiplier Methods, *Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 111, pp. 73–80.

Pagalday, J.M. (1994) *Optimización del comportamiento dinámico de mecanismos* (Doctoral Thesis), Universidad de Navarra.

Pagalday, J.M. and Avello, A. (1997) Optimization of multibody dynamics using object oriented programming and a mixed numerical-symbolic penalty formulation, *Mech. Mach. Theory*, vol. 32, no. 2, pp. 161–174.

Pasquotti, M. (2008) *Topics on Optimization Strategies for Constrained Mechanical Systems* (Doctoral Thesis), Università di Padova.

Pi, T., Zhang, Y. and Chen, L. (2012) First order sensitivity analysis of flexible multibody systems using absolute nodal coordinate formulation, *Multibody System Dynamics*, vol. 27, pp. 153–171.

Polach, P. and Hajžman, M. (2008) Design of characteristics of air-pressure-controlled hydraulic shock absorbers in an intercity bus, *Multibody System Dynamics*, vol. 19, pp. 73–90.

Rall, L.B. (1981) Automatic Differentiation: Techniques and Applications, *Lecture Notes in Computer Science*, vol. 120.

Roberson, R.E. and Schwertassek, R. (1988) *Dynamics of Multibody Systems*, Springer-Verlag.

Rodríguez, J.I., Jiménez, J.M., Funes, F.J. and García de Jalón, J. (2004) Recursive and Residual Algorithms for the Efficient Numerical Integration of Multi-Body Systems, *Multibody System Dynamics*, vol. 11, pp. 295–320.

Schaffer, A.S. (2005) *On the adjoint formulation of design sensitivity analysis of multibody dynamics* (Doctoral Thesis), University of Iowa.

Schiehlen, W. (1990) *Multibody Systems Handbook*, Springer-Verlag.

Schiehlen, W. (1997) Multibody System Dynamics: Root and Perspectives, *Multibody System Dynamics*, vol. 1, pp. 149–188.

Schordan, M. and Quinlan, D. (2003) A source-to-source architecture for userdefined optimizations, in *Lecture Notes in Computer Science*, Springer Verlag.

Serban, R. and Freeman, J.S. (1996) Direct differentiation methods for the design sensitivity of multibody dynamic systems, The 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference.

Serban, R. and Freeman, J.S. (2001) Identification and Identifiability of Unknown Parameters in Multibody Dynamic Systems, *Multibody System Dynamics*, vol. 5, pp. 335–350.

Serban, R. and Haug, E.J. (1998) Kinematic and Kinetic Derivatives in Multibody System Analysis, *Mech. Struct. & Mach.*, vol. 26, no. 2, pp. 145–173.

Serban, R. and Haug, E.J. (2000) Globally Independent Coordinates for Real-Time Vehicle Simulation, *ASME Journal of Mechanical Design*, vol. 122, pp. 575–582.

Serban, R., Negrut, D., Potra, F.A. and Haug, E.J. (1997) A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics. Cartesian Formulation, *Mechanics of Structures and Machines*, vol. 25, no. 3.

Serna, M.A., Avilés, R. and García de Jalón, J. (1982) Dynamic Analysis of Plane Mechanisms with Lower Pairs in Basic Coordinates, *Mechanisms and Machine Theory*, vol. 17, pp. 397–403.

Shabana, A.A. (1989) *Dynamics of Multibody Systems*, Wiley.

Siskind, J.M. and Pearlmutter, B.A. (2008) *Putting the Automatic Back into AD*, Purdue University.

Sohn, J.-H., Park, S.-J., Lee, J.-H., Choi, S.-H. and Yoo, W.-S. (2010) Full Vehicle Simulation for Durability Test of Damper in a Cruise Bus, The 1st Joint International Conference on Multibody System Dynamics, Lappeenranta.

Sohoni, V.N. and Haug, E.J. (1982) A state space technique for optimal design of mechanisms, *ASME Journal of Mechanical Design*, vol. 104, pp. 792–798.

Sonneville, V. and Brüls, O. (2013) Sensitivity analysis for multibody systems formulated on a Lie group, *Multibody System Dynamics*, pp. 1–21.

Stadler, W. and Eberhard, P. (2001) Jacobian motion and its derivatives, *Mechatronics*, vol. 11, pp. 563–593.

Stolpe, M. and Kawamoto, A. (2005) Design of planar articulated mechanisms using branch and bound, *Mathematical Programming*, vol. 103, pp. 357–397.

Strout, M.M., Mellor-Crummey, J. and Hovland, P. (2006) Representation-independent program analysis, *SIGSOFT Softw. Eng. Notes*, vol. 31, pp. 67–74.

Thoresson, M.J. (2007) *Efficient Gradient-Based Optimisation of Suspension Characteristics for an Off-Road Vehicle* (Doctoral Thesis), University of Pretoria.

Thoresson, M.J., Uys, P.E., Els, P.S. and Snyman, J.A. (2009) Efficient optimisation of a vehicle suspension system, using a gradient-based approximation method, Part 1: Mathematical modelling, *Mathematical and Computer Modelling*, vol. 50, pp. 1421–1436.

Tomovic, R. (1963) *Sensitivity Analysis of Dynamic Systems*, New York: McGraw-Hill.

Utke, J. (2004) *OpenAD: Algorithm Implementation User Guide*, Mathematics and Computer Science Division, Argonne National Laboratory.

Venkataraman, P. (2002) *Applied Optimization with MATLAB Programming*, New York: John Wiley & Sons.

Vereschagin, A. (1974) Computer Simulation of the Dynamics of Complicated Mechanisms of Robot-Manipulators, *Engineering Cybernetics*, vol. 6, pp. 65–70.

Vidal, J. (2006) *Un Método General, Sencillo y Eficiente, para la Definición y Simulación Numérica de Sistemas Multicuerpo* (Doctoral Thesis), Madrid: Universidad Politécnica de Madrid.

Von Schwerin, R. (1999) *Multibody System Simulation. Numerical Methods, Algorithms and Software*, Springer.

Walker, M.W. and Orin, D.E. (1982) Efficient Dynamic Computer Simulation of Robotic Mechanisms, *ASME Journal of Dynamic Systems, Measurements and Control*, vol. 104, pp. 205–211.

Walther, A. and Griewank, A. (2012) *ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++. Version 2.3.0*.

Wang, X., Haug, E.J. and Pan, W. (2005) Implicit Numerical Integration for Design Sensitivity Analysis of Rigid Multibody Systems, *Mechanics Based Design of Structures and Machines*, vol. 33, pp. 1–30.

Wehage, R.A. and Haug, E.J. (1982) Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Dynamic Systems, *ASME Journal of Mechanical Design*, vol. 104, pp. 247–255.

Wittenburg, J. (1977) *Dynamics of Systems of Rigid Bodies*, Teubner.

Appendix A

Platform and software characteristics

Computer	Processor: Intel Core i7 870 @ 2.93 GHz Memory: 6 GB GPU: NVIDIA GTS 240 1024 MB OS I: 64-bit Windows 7 Professional (SP1) OS II: 32-bit Ubuntu 12.04
MATLAB	2011a (64-bit) 7.12.0.635
Microsoft Visual Studio	2008 (9.0)
MSC Adams	2012 (64-bit)
ADOL-C	2.3.0
ADIC2	(ADIC) 2.0

Appendix B

Roll center

An analysis of the roll axis is useful to understand the roll behavior of the coach. The roll axis is defined as the axis around which the vehicle rolls when side forces and inertias appear. Ideally, if one applies a horizontal side force on a point of the bodywork roll axis, the bodywork does not roll. Usually the roll axis is in the XZ-plane. In the case of the coach, the front roll center is on the floor because the suspension triangles are almost horizontal (see Figure 7.1) and therefore their planes intersect at infinity. On the other hand, the rear roll center can be easily calculated graphically by following the four steps depicted in Figure 7.2, where plan and side views of the rear axle are shown. The variation of the roll centers with the vehicle movement is neglected in this analysis.

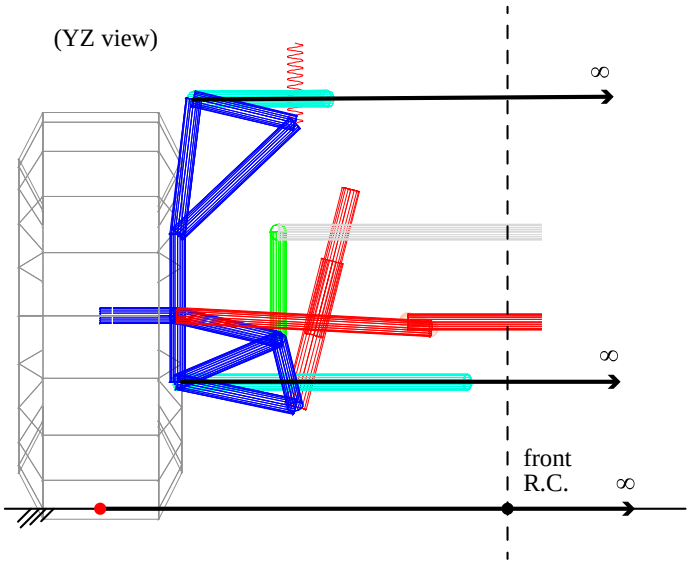


Figure 7.1: Graphic calculation of the front roll center.

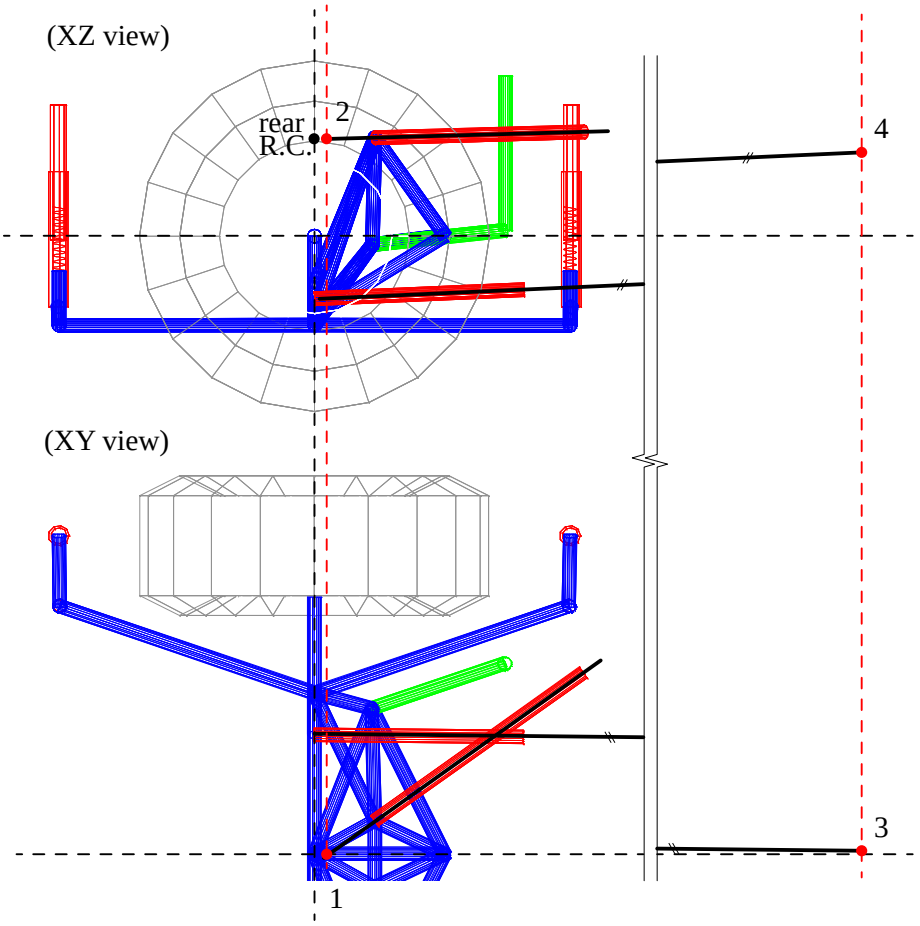


Figure 7.2: Graphic calculation of the rear roll center.

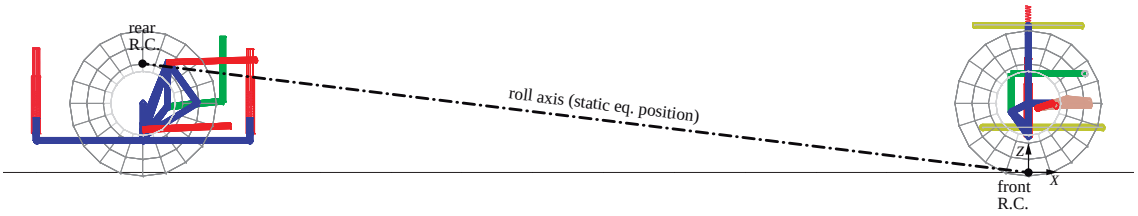


Figure 7.3: Roll axis in the static equilibrium position.

Appendix C

Tire parameters

Scaling coefficients		**Longitudinal coefficients**	
Coefficient	Value	Coefficient	Value
LFZO	1	PCX1	1.7204
LCX	1	PDX1	0.77751
LMUX	1	PDX2	–0.24431
LEX	1	PDX3	–0.00015908
LKX	1	PEX1	0.46659
LHX	1	PEX2	0.393
LVX	1	PEX3	0.076024
LGAX	1	PEX4	2.65×10^{-6}
LCY	1	PKX1	14.848
LMUY	1	PKX2	–9.8161
LEY	1	PKX3	0.15818
LKY	1	PHX1	–0.00088873
LHY	1	PHX2	–0.00067818
LVY	1	PVX1	-5.57×10^{-7}
LGAY	1	PVX2	6.30×10^{-6}
LTR	1	RBX1	11.13
LRES	1	RBX2	–12.494
LGAZ	1	RCX1	0.97505
LXAL	1	REX1	–0.37196
LYKA	1	REX2	0.0017379
LVYKA	1	RHX1	0.0045181
LS	1	PTX1	1.5
LSGKP	1	PTX2	1.4
LSGAL	1	PTX3	1
LGYR	1		
LMX	1		
LVMX	1		
LMY	1		

Overturning coefficients		**Rolling coefficients**	
Coefficient	Value	Coefficient	Value
QSX1	0	QSY1	0.008
QSX2	0	QSY2	0
QSX3	0	QSY3	0
		QSY4	0

Lateral coefficients		**Aligning coefficients**	
Coefficient	Value	Coefficient	Value
PCY1	1.5874	QBZ1	5.8978
PDY1	0.73957	QBZ2	–0.1535
PDY2	–0.075004	QBZ3	–2.0052
PDY3	–8.0362	QBZ4	0.62731
PEY1	0.37562	QBZ5	–0.92709
PEY2	–0.069325	QBZ9	10.637
PEY3	0.29168	QBZ10	0
PEY4	11.559	QCZ1	1.4982
PKY1	–10.289	QDZ1	0.085549
PKY2	3.3343	QDZ2	–0.025298
PKY3	–0.25732	QDZ3	0.21486
PHY1	0.0056509	QDZ4	–3.9098
PHY2	–0.0020257	QDZ6	–0.0013373
PHY3	–0.038716	QDZ7	0.0013869
PVY1	0.015216	QDZ8	–0.053513
PVY2	–0.010365	QDZ9	0.025817
PVY3	–0.31373	QEZ1	–0.0084519
PVY4	–0.055766	QEZ2	0.0097389
RBY1	13.271	QEZ3	0
RBY2	5.2405	QEZ4	4.3583
RBY3	1.15×10^{-5}	QEZ5	–645.04
RCY1	1.01	QHZ1	0.0085657
REY1	0.010513	QHZ2	–0.0042922
REY2	5.98×10^{-5}	QHZ3	0.14763
RHY1	0.028005	QHZ4	–0.29999
RHY2	-4.88×10^{-5}	SSZ1	–0.019408
RVY1	0.0066878	SSZ2	0.025786
RVY2	–0.042813	SSZ3	0.31908
RVY3	–0.16227	SSZ4	–0.50765
RVY4	–0.019796	QTZ1	0
RVY5	1.9	MBELT	0
RVY6	–7.8097		
PTY1	1.2		
PTY2	2.5		